

DESIGN AND ANALYSIS OF EFFICIENT AND SECURE ELLIPTIC CURVE CRYPTOPROCESSORS

BY

TURKI FAISAL AL-SOMANI

A Dissertation Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

In

COMPUTER SCIENCE AND ENGINEERING

May, 2006

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

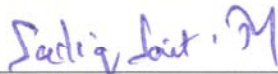
DHAHRAN 31261, SAUDI ARABIA


DEANSHIP OF GRADUATE STUDIES


This dissertation, written by **TURKI FAISAL AL-SOMANI** under the direction of his dissertation advisor and approved by his dissertation committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE AND ENGINEERING**.


Dissertation Committee

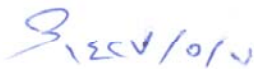

Dr. Alaaeldin A. Amin
Dissertation Committee Chairman


Prof. Sadiq Sait Mohammed
Member


Dr. Mohammad H. Al - Suwaiyel
Member


Dr. Adnan A. Gutub
Department Chairman


Dr. Mohammad A. Al - Ohali
Dean of Graduate Studies


Date 3.6.2006



Dedication

To my beloved wife Amal .. for her love and patience.

Acknowledgments

All praise and thanks be to Almighty Allah, the most Gracious, the most Merciful. Peace and mercy be upon His Prophet.

Acknowledgment is due to the King Fahd University of Petroleum & Minerals (KFUPM) and Umm Al-Qura University (UQU) for supporting this research. I would like also to acknowledge the support of King Abdul Aziz City for Science and Technology (KACST) for the research grant No. SR-13-81.

I am eternally grateful to my supervisor Dr. Alaaeldin A. Amin for his constant support and many suggestions without who I would not be able to successfully complete this research work. Many thanks are extended to my committee members Prof. Sadiq Sait Mohammed and Dr. Mohammad H. Al-Suwaiyel for their unlimited encouragement and useful suggestions.

I am also thankful to Dr. Mohammad K. Ibrahim for his guidance at the beginning of this research. I also would like to thank Dr. Aiman H. El-Maleh and Dr. Muhammad E. Elrabaa for their valuable discussions.

I express my gratitude to Dr. Jarallah S. Al-Ghamdi, Dean of College of Computer Sciences & Engineering, Dr. Adnan A. Gutub, Chairman of Computer Engineering Department, and Dr. Kanaan A. Faisal, Chairman of Information & Computer Science Department for their support.

I am thankful for all the faculty and staff members of Computer Engineering

Department and Information & Computer Science Department for their support.

I would also acknowledge the help of Dr. Mahmoud Kassas, Mr. Noman Tasad-
duq, and Mr. Umar Johar from Electrical Engineering Department for their help in
lab assistance.

I am grateful to Mr. Khalid O. Al-Summani from Ministry of Education, Dr.
Ali Al-Ogla from UQU, and Eng. Mahboob Al-Mahboob from KACST for their
continuous encouragement and support.

I would like to acknowledge my colleagues and friends especially Mr. Esa Al-
Ghonaim, Mr. Salman Al-Qahtani, Mr. Theeb Al-Gahtani, and Mr. Abdul-Aziz
Al-Khoraidly for their encouragement.

Finally, I am grateful to my parents and family for their moral support, encour-
agement and prayers without which this work would not have been possible.

Turki F. Al-Somani.

MAY, 2006.

Contents

Acknowledgments	ii
List of Tables	vii
List of Figures	viii
List of Algorithms	ix
Abstract (English)	x
Abstract (Arabic)	xi
1 Introduction	1
1.1 Motivation	1
1.2 Organization of the Dissertation	3
2 Background	5
2.1 Finite Field Arithmetic	5
2.2 $GF(2^m)$ Arithmetic	7
2.3 Elliptic Curve Arithmetic	11
2.4 Scalar Multiplication	16
2.5 Elliptic Curve Encryption	19
2.5.1 Elliptic Curve Diffie-Hellman Protocol	19
2.5.2 Elliptic Curve ElGamal Protocol	20
2.6 Elliptic Curve Discrete Logarithm Problem	20
2.7 Summary	21
3 Scalar Multiplication Algorithms	23
3.1 Fixed Space-Time Scalar Multiplication Algorithms	23
3.1.1 Double-and-Add Scalar Multiplication Algorithm	24
3.1.2 Addition-Subtraction Scalar Multiplication Algorithm	25
3.1.3 Montgomery Scalar Multiplication Algorithm	27

3.2	Flexible Space-Time Scalar Multiplication Algorithms	28
3.2.1	w -ary Scalar Multiplication Algorithm	28
3.2.2	w -ary Addition-subtraction Scalar Multiplication Algorithm .	29
3.2.3	Width- w Addition-Subtraction Scalar Multiplication Algorithm	31
3.2.4	Signed BGMW Scalar Multiplication Algorithm	33
3.2.5	Lim-Lee Scalar Multiplication Algorithm	35
3.3	Summary	37
4	Normal Basis $GF(2^m)$ Field Arithmetic	40
4.1	Multiplication	41
4.1.1	λ -Matrix Based Multipliers	41
4.1.2	Conversion Based Multipliers	46
4.2	Inversion	51
4.2.1	Standard Inversion Algorithm	51
4.2.2	Exponent Decomposing Inversion Algorithms	52
4.2.3	Exponent Grouping Inversion Algorithms	55
4.3	Summary	57
5	Power Analysis Attacks and $GF(2^m)$ FPGA Implementations	60
5.1	Power Analysis Attacks	60
5.1.1	Simple Power Analysis	61
5.1.2	Differential Power Analysis	62
5.2	$GF(2^m)$ FPGA Implementations	67
5.3	Remarks on the reviewed implementations	72
5.4	Summary	75
6	Secure ECC Cryptoprocessor Architectures	76
6.1	The ECC_{NS} Cryptoprocessor	77
6.1.1	Main Controller	78
6.1.2	Data Embedding	78
6.1.3	Point Addition and Doubling	81
6.1.4	Field Operations	82
6.2	The ECC_{SS} Cryptoprocessor	95
6.2.1	Key Partitioning	96
6.2.2	Multilevel Resistance Measures	99
6.2.3	ECC_{SS} Architecture and Operation	107
6.3	The ECC_{PS} Cryptoprocessor	108
6.3.1	ECC_{PS} Architecture and Operation	115
6.3.2	Security, Space and Time Analysis	116
6.4	Summary	117

7	Results and Discussions	121
7.1	The ECC_{NS} Cryptoprocessor	122
7.2	The ECC_{SS} Cryptoprocessor	124
7.3	The ECC_{PS} Cryptoprocessor	128
7.4	ECC_{SS} vs. ECC_{PS} Comparison	130
7.5	Summary	133
8	Conclusions and Future Research	137
	BIBLIOGRAPHY	140

List of Tables

2.1	The Homogeneous projective coordinate system.	16
2.2	The Jacobian projective coordinate system.	17
2.3	The Lopez-Dahab projective coordinate system.	18
3.1	Complexity of scalar multiplication algorithms	38
4.1	The λ -based multipliers and their space and time complexities.	45
4.2	The conversion based multipliers and their space and time complexities.	50
4.3	The inverters and their time complexities.	58
5.1	$GF(2^m)$ Implementations on FPGAs.	74
6.1	Lopez-Dahab Projective Coordinate System.	94
7.1	The ECC_{NS} Cryptoprocessor Synthesis Results.	123
7.2	The ECC_{SS} Cryptoprocessor Synthesis Results.	126
7.3	The ECC_{PS} Cryptoprocessor Synthesis Results.	129
7.4	Area-Time Complexity Comparison for $u = 4$	133

List of Figures

2.1	The point addition operation ($R = P + Q$) over $GF(p)$	13
2.2	The point doubling operation ($R = 2P$) over $GF(p)$	13
4.1	$GF(2^5)$ bit-serial Massey-Omura multiplier [77].	42
4.2	Gao and Sobelman multiplier [80].	44
4.3	The Wu <i>et. al.</i> multiplier [85].	50
5.1	Gao <i>et. al.</i> ECC coprocessor [22].	68
5.2	ECC processor architecture used by Leung <i>et. al.</i> [25] and Leong <i>et. al.</i> [30]	69
5.3	Ernst <i>et. al.</i> ECC architecture [27].	70
5.4	Bednara <i>et. al.</i> ECC architecture [36].	71
5.5	Cheung <i>et. al.</i> ECC architecture [45].	72
5.6	Al-Somani and Ibrahim ECC architecture [47].	73
6.1	The proposed architecture	77
6.2	Dataflow of the proposed sequential multiplier.	87
6.3	Circuit of $C1$, $i = [1, m - 1]$, $j = [2, m]$	89
6.4	Circuit of $D1$, $i = [1, m - 1]$, $j = [1, m - 1]$	90
6.5	Circuit of $D2$, $i = [1, m]$, $j = [1, m]$	91
6.6	Dataflow of the Itoh and Tsujii inverter [87].	92
6.7	Multilevel resistance measures.	104
6.8	The proposed parallel architecture.	112
6.9	Key partitioning and parallel execution.	114
7.1	The ECC_{SS} Cryptoprocessor Delay and Area Overheads.	127
7.2	The ECC_{PS} Cryptoprocessor Delay and Area Overheads.	130
7.3	The ECC_{PS} Cryptoprocessor Voltage Trace with $m = 11$	131
7.4	The Area-Time Complexity for $u = 4$	133
7.5	The Area-Time ² Complexity for $u = 4$	134
7.6	The Approximated Area-Time Complexity for $m = 173$	135
7.7	The Approximated Area-Time ² Complexity for $m = 173$	136

List of Algorithms

3.1	Double-and-add scalar multiplication algorithm (most-to-least).	24
3.2	Double-and-add scalar multiplication algorithm (least-to-most).	25
3.3	Addition-subtraction scalar multiplication algorithm.	26
3.4	Montgomery Scalar Multiplication Algorithm.	27
3.5	w -ary scalar multiplication algorithm.	29
3.6	w -ary addition-subtraction scalar multiplication algorithm.	30
3.7	Width- w Addition-Subtraction Scalar Multiplication Algorithm.	32
3.8	Signed BGMW Scalar Multiplication Algorithm.	34
3.9	Lim-Lee Scalar Multiplication Algorithm.	36
4.1	Wang's <i>et. al.</i> inversion algorithm.	51
4.2	Itoh-Tsujii inversion algorithm.	53
4.3	Feng's inversion algorithm.	54
5.1	Double-and-add-always Scalar Multiplication Algorithm.	62
6.1	Pseudocode of the ECC_{NS} cryptoprocessor.	79
6.2	Pseudocode of the conversion box unit.	85
6.3	Pseudocode of \widehat{C}_1 .	89
6.4	Pseudocode of \widehat{D}_1 .	90
6.5	Pseudocode of \widehat{D}_2 .	91
6.6	Itoh-Tsujii inversion algorithm.	93
6.7	NAF encoding algorithm.	100
6.8	The Randomized bit Algorithm (most-to-least)	102
6.9	The Randomized bit Algorithm (least-to-most)	102
6.10	Pseudocode of the ECC_{SS} cryptoprocessor.	109
6.11	Pseudocode of the ECC_{PS} cryptoprocessor.	119
6.12	Modified double-and-add scalar multiplication algorithm (most-to-least).	120
6.13	Modified double-and-add scalar multiplication algorithm (least-to-most).	120

DISSERTATION ABSTRACT

Name: TURKI FAISAL AL-SOMANI

Title: DESIGN AND ANALYSIS OF EFFICIENT
AND SECURE ELLIPTIC CURVE
CRYPTOPROCESSORS

Major Field: COMPUTER SCIENCE AND ENGINEERING

Date of Degree: MAY 2006

Elliptic Curve Cryptosystems have attracted many researchers and have been included in many standards such as IEEE, ANSI, NIST, SEC and WTLS. The ability to use smaller keys and computationally more efficient algorithms compared with earlier public key cryptosystems such as RSA and ElGamal are two main reasons why elliptic curve cryptosystems are becoming more popular. They are considered to be particularly suitable for implementation on smart cards or mobile devices. Power Analysis Attacks on such devices are considered serious threat due to the physical characteristics of these devices and their use in potentially hostile environments.

This dissertation investigates elliptic curve cryptoprocessor architectures for curves defined over $GF(2^m)$ fields. In this dissertation, new architectures that are suitable for efficient computation of scalar multiplications with resistance against power analysis attacks are proposed and their performance evaluated. This is achieved by exploiting parallelism and randomized processing techniques. Parallelism and randomization are controlled at different levels to provide more efficiency and security. Furthermore, the proposed architectures are flexible enough to allow designers tailor performance and hardware requirements according to their performance and cost objectives. The proposed architectures have been modeled using VHDL and implemented on FPGA platform.

DOCTOR OF PHILOSOPHY DEGREE

King Fahd University of Petroleum & Minerals, Dhahran.

MAY 2006

ملخص الأطروحة

الإسم : تركي فيصل عبيد الصماني
العنوان : تصميم وتحليل معالجات تشفير فعّالة وأمنة على المنحنيات البيضاوية
التخصص : علوم وهندسة الحاسب الآلي
تاريخ الدرجة : ربيع الثاني 1427 هـ

لقد جذبت أنظمة التشفير المبنية على المنحنيات البيضاوية اهتمام الكثير من الباحثين ، كما أنها وُضعت في كثير من المواصفات القياسية. إن استخدام مفاتيح ذات أطوال قصيرة بالإضافة إلى استخدام خوارزميات أكثر فعالية من المستخدمة سابقاً في أنظمة التشفير ذات المفتاح العمومي هما الأسباب الرئيسة جعل أنظمة التشفير المبنية على المنحنيات البيضاوية أكثر شعبية. تعد هذه الأنظمة مناسبة للتنفيذ على البطاقات الذكية والأجهزة النقالة ، ولكن الهجمات باستخدام تحليل القدرة على مثل هذه الأجهزة يشكل خطراً حقيقياً عند استخدامها في بيئات غير آمنة ، كما أن عدم الاهتمام بهذه الهجمات قد يجعل المفاتيح السرية قابلة للاستنتاج أو التخمين. تقدم رسالة الدكتوراه هذه بنية معالجات على المنحنيات البيضاوية فعّالة وأمنة ضد هجمات تحليل القدرة ، وتقوم باقتراح بنى لمعالجات ذات كفاءة عالية في حساب عملية ضرب النقاط على المنحنيات البيضاوية ، وذلك من خلال استثمار البنى المتوازية والبنى العشوائية القابلة لإعادة إعداد البنية ، علماً بأن التحكم في التوازي والعشوائية هو على مستوى خوارزميات ضرب النقاط على المنحنيات البيضاوية ، كما أن بنى المعالجات المقترحة تعطي المصممين الحرية في تقنين الأداء ومتطلبات الأجهزة والمعدات وفقاً لتكلفة الأهداف .

1427

Chapter 1

Introduction

1.1 Motivation

Elliptic Curve Cryptosystems (ECCs) [1] have been recently attracting increased attention. Standards for ECCs have been adopted by IEEE, ANSI, NIST, SEC and WTLS [2]–[8]. The ability to use smaller key sizes and the computationally more efficient ECC algorithms are two main reasons why elliptic curve cryptosystems are becoming more popular. They are considered to be particularly suitable for implementation on platforms with constrained storage and/or battery specifications, e.g. smart cards or mobile devices.

Power analysis attacks [11] on such devices are considered serious threats due to the physical characteristics of these devices and their use in potentially hostile environments. Power analysis attacks seek to break the security of these devices

through observing their power consumption trace or computations timing. Careless or naive implementations of cryptosystems may allow power analysis attacks to infer the secret key or obtain partial information about it. Thus, designers of such systems strive to introduce algorithms and architectures that are not only efficient, but also power analysis attack resistant.

Several software implementations of elliptic curve cryptosystems have been reported [13]–[18]. The advantages of software implementations include ease of use, ease of upgrade, portability, low development cost and flexibility. Their main disadvantages, on the other hand, are their lower performance and limited ability to protect private keys from disclosure compared to hardware implementations. These disadvantages have motivated many researchers to investigate efficient architectures for hardware implementations of elliptic curve cryptosystems.

Several hardware implementations of elliptic curve cryptosystems have been reported. Most proposed hardware architectures were for elliptic curves cryptosystems defined over $GF(2^m)$ [19]–[47]. Many elliptic curve implementations over $GF(p)$ have also been reported, e.g., [29, 36, 37, 50–54]. Hardware implementations offer improved speed and higher security over software implementations, because they cannot be read or modified by an outside attacker as easily as software implementations.

Application Specific Integrated Circuits (ASIC) implementations show lower price per unit, high speeds, and low power dissipation. The main disadvantages

of ASIC implementations, however, are higher development costs and the lack of flexibility. Field Programmable Gate Array (FPGA) technology offers a good compromise between the speed of ASIC implementations, the short development times, and adaptability of software implementations.

In this research, efficient elliptic curve cryptoprocessor architectures that are resistant to known power analysis attacks have been developed. The proposed architectures exploit parallelism and randomization to provide high performance and resistance against power analysis attacks. These proposed architectures have been modeled in VHSIC Hardware Description Language (VHDL) and implemented on an FPGAs platform.

1.2 Organization of the Dissertation

This chapter provides the motivation for the work performed in this dissertation.

The remaining chapters of this dissertation are organized as follows.

Chapter 2 provides a brief introduction to $GF(2^m)$ finite field arithmetic. ECC operations including scalar multiplication, encryption and discrete logarithm problem are also briefly explored in Chapter 2.

Chapter 3 reviews some of the most popular scalar multiplication algorithms. Normal basis multiplication and inversion algorithms over $GF(2^m)$ are explained in Chapter 4.

Chapter 5 surveys techniques for power analysis attacks and FPGA implementations of ECCs. The proposed power analysis attack-resistant cryptoprocessors are detailed in Chapter 6.

Chapter 7 presents the results of synthesizing the various cryptoprocessors and compares these cryptoprocessors in terms of delay and area. Finally, the conclusions and future work are given in Chapter 8.

Chapter 2

Background

This chapter provides a brief introduction to $GF(2^m)$ finite field arithmetic. ECC operations including scalar multiplication, encryption and discrete logarithm problem are also briefly explored in this chapter.

2.1 Finite Field Arithmetic

In abstract algebra, a *finite field* is a field that contains only finitely many elements. Finite fields are important in number theory, algebraic geometry, Galois theory, coding theory, and cryptography [55]–[57].

A *group* is a set of elements G together with one binary operation, \diamond , which have the following properties:

1. **Closure:** $\forall a, b \in G, a \diamond b \in G$.

2. **Associativity:** $\forall a, b, c \in G, (a \diamond b) \diamond c = a \diamond (b \diamond c)$.

3. **Identity:** The group contains an identity element $e \in G$ such that

$$\forall a \in G, a \diamond e = e \diamond a = a.$$

4. **Inverse:** Every element $a \in G$ has an inverse $a^{-1} \in G$ such that $a \diamond a^{-1} =$

$$a^{-1} \diamond a = e.$$

Abelian groups are groups with commutative group operation; i.e., $a \diamond b = b \diamond a$ $\forall a, b \in G$. Cyclic groups are groups that have a generator element. An element $g \in G$, is a generator of the group if each element $a \in G$ can be generated by repeated application of the group operation on g . Thus, $\forall a \in G$,

$$a = \underbrace{g \diamond g \diamond g \diamond \dots \diamond g}_{i \text{ times}} \quad (2.1)$$

Additive groups, are groups with the “+” group operator, denoted as:

$$ig = \underbrace{g + g + g + \dots + g}_{i \text{ times}} \quad (2.2)$$

Similarly, *multiplicative groups*, are groups with the “*” group operator, denoted as:

$$g^i = \underbrace{g * g * g * \dots * g}_{i \text{ times}} \quad (2.3)$$

The order of a group G , represented by the symbol $|G|$, is the number of elements

in the group.

A *field* is a set of elements F with two binary operations, represented here as addition “+” and multiplication “*”, which have the following properties:

1. F is an abelian group with respect to the “+” operation.
2. The elements of the set F^* form an abelian group under the “*” operation.

The set F^* is a set that contains all the elements in F except the additive identity.

3. The distribution law applies to the two binary operations as follows:

$$\forall a, b, c \in F, a * (b + c) = (a * b) + (a * c).$$

Finite fields or Galois field, so named in honor of Evariste Galois, are represented by the symbol $GF(q)$. For any prime p and positive integer m , there always exists a Galois field of order $q = p^m$. The prime p is called the characteristic of the finite field $GF(p^m)$.

2.2 $GF(2^m)$ Arithmetic

The finite $GF(2^m)$ field, with characteristic 2, has particular importance in cryptography since it leads to efficient hardware implementations. Elements of the $GF(2^m)$

field are represented in terms of a basis. Most implementations use either a *Polynomial Basis* or a *Normal Basis*. For the implementations described in this dissertation, a normal basis is chosen since it leads to more efficient hardware. Normal basis is more suitable for hardware implementations than polynomial basis since operations mainly comprise rotation, shifting and exclusive-ORing which can be efficiently implemented in hardware. A normal basis of $GF(2^m)$ is a basis of the form $(\beta^{2^{m-1}}, \dots, \beta^{2^2}, \beta^{2^1}, \beta^{2^0})$, where $\beta \in GF(2^m)$.

In a normal basis, an element $A \in GF(2^m)$ can be uniquely represented in the form $A = \sum_{i=0}^{m-1} \alpha_i \beta^{2^i}$, where $\alpha_i \in \{0, 1\}$. $GF(2^m)$ operations using normal basis are performed as follows:

1. **Addition.** Addition is performed by a simple bit-wise exclusive-OR (XOR) operation.

2. **Squaring.** Squaring is simply a rotate left operation. Thus, if

$$A = (a_{m-1}, a_{m-2}, \dots, a_1, a_0), \text{ then } A^2 = (a_{m-2}, a_{m-3}, \dots, a_0, a_{m-1}).$$

3. **Multiplication.** $\forall A, B \in GF(2^m)$, where

$$A = \sum_{i=0}^{m-1} a_i \beta^{2^i} \text{ and } B = \sum_{i=0}^{m-1} b_i \beta^{2^i},$$

the product $C = A * B$, is given by:

$$C = A * B = \sum_{i=0}^{m-1} c_i \beta^{2^i}$$

Multiplication is defined in terms of a set of m multiplication matrices $\lambda^{(k)}$ ($k = 0, 1, \dots, m - 1$),

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \lambda_{ij}^{(k)} a_i b_j \quad \forall k = 0, 1, \dots, m - 1$$

where, $\lambda_{ij}^{(k)} \in \{0, 1\}$.

The number of non-zero elements in the λ matrix defines the complexity of the multiplication process and accordingly the complexity of its hardware implementation. This value is denoted as C_N and is equal to $(2m - 1)$ for optimal normal basis (ONB) [58]. An optimal normal basis is one with the minimum possible number of non-zero elements in the λ_{ij} matrix. Such bases typically lead to efficient hardware implementations since operations mainly comprise rotation, shifting and exclusive-ORing.

Derivation of values of the λ matrix elements is dependent on the field size m . There are two types of optimal normal basis that are referred to as Type I and Type II [58]. An ONB of Type I exists for a given field $GF(2^m)$ if:

- (a) $m + 1$ is a prime
- (b) 2 is a primitive in $GF(m + 1)$

On the other hand, a Type II optimal normal basis exists in $GF(2^m)$ if:

- (a) $2m + 1$ is prime

- (b) either 2 is a primitive in $GF(2m+1)$ or $2m+1 \equiv 3 \pmod{4}$ and 2 generates the quadratic residues in $GF(2m+1)$

An ONB exists in $GF(2^m)$ for 23% of all possible values of m [58]. The $\lambda^{(k)}$ matrix can be constructed by a k -fold cyclic shift to $\lambda^{(0)}$ as follows:

$$\lambda_{ij}^{(k)} = \lambda_{i-k, j-k}^{(0)} \text{ for all } 0 \leq i, j, k \leq m-1$$

The $\lambda^{(0)}$ matrix is derived differently for the two types of ONB. For the Type I ONB, $\lambda_{ij}^{(0)} = 1$ iff i and j satisfy one of the following two congruences [59]:

- (a) $2^i + 2^j \equiv 1 \pmod{m+1}$
- (b) $2^i + 2^j \equiv 0 \pmod{m+1}$

For Type II ONB, $\lambda_{ij}^{(k)} = 1$ iff i and j satisfy one of the following four congruences [59]:

- (a) $2^i + 2^j \equiv 2^k \pmod{2m+1}$
- (b) $2^i + 2^j \equiv -2^k \pmod{2m+1}$
- (c) $2^i - 2^j \equiv 2^k \pmod{2m+1}$
- (d) $2^i - 2^j \equiv -2^k \pmod{2m+1}$

Therefore, $\lambda_{ij}^{(0)} = 1$ iff i and j satisfy one of the following four congruences:

$$2^i \pm 2^j \equiv \pm 1 \pmod{2m+1}$$

4. **Inversion.** Inverse of $a \in GF(2^m)$, denoted as a^{-1} , is defined as follows.

$$aa^{-1} \equiv 1 \text{ mod } 2^m$$

Most inversion algorithms are derived from Fermat's Little Theorem, where

$$a^{-1} = a^{2^m-2}$$

for all $a \neq 0$ in $GF(2^m)$.

2.3 Elliptic Curve Arithmetic

An elliptic curve E over the finite field $GF(p)$ defined by the parameters $a, b \in GF(p)$ with $p > 3$, consists of the set of points $P = (x, y)$, where $x, y \in GF(p)$, that satisfy the elliptic curve equation (Equation 2.4) together with the additive identity of the group point \mathcal{O} , known as the “point at infinity” [1].

$$y^2 = x^3 + ax + b \tag{2.4}$$

where $a, b \in GF(p)$ and $4a^3 + 27b^2 \neq 0 \text{ mod } p$.

The number of points n on an elliptic curve over a finite field $GF(q)$ is defined by Hasse's theorem [56]. The set of discrete points on an elliptic curve form an abelian group, whose group operation is known as point addition. Elliptic curve

point addition is defined according to the “chord-tangent process”. Point addition over $GF(p)$ is described as follows.

Let P and Q be two distinct points on E defined over $GF(p)$ with $Q \neq -P$ (Q is not the additive inverse of P). The addition of the two points P and Q is the point R ($R = P + Q$), where R is the additive inverse of S , and S is a third point on E intercepted by the straight line through points P and Q . The additive inverse of a point $P = (x, y) \in E$, over $GF(p)$, is the point $-P = (x, -y)$ which is the reflection of the point P with respect to the x -axis on E . The addition operation over $GF(p)$ is depicted in Figure 2.1.

When $P = Q$ and $P \neq -P$, the addition of P and Q is the point R ($R = 2P$), where R is the additive inverse of S , and S is the third point on E intercepted by the straight line tangent to the curve at point P . This operation is referred to as *point doubling*, and is shown in Figure 2.2.

Equation 2.5 defines the non-supersingular elliptic curve equation for $GF(2^m)$ fields. Only non-supersingular curves over $GF(2^m)$ are considered since supersingular curves are not secure. Supersingular elliptic curves are special class of curves with some special properties that make them unstable for cryptography [60].

$$y^2 + xy = x^3 + ax^2 + b \tag{2.5}$$

where $a, b \in GF(2^m)$ and $b \neq 0$.

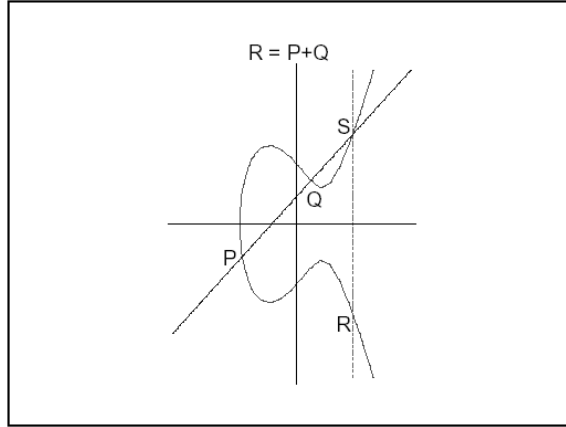


Figure 2.1: The point addition operation ($R = P + Q$) over $GF(p)$.

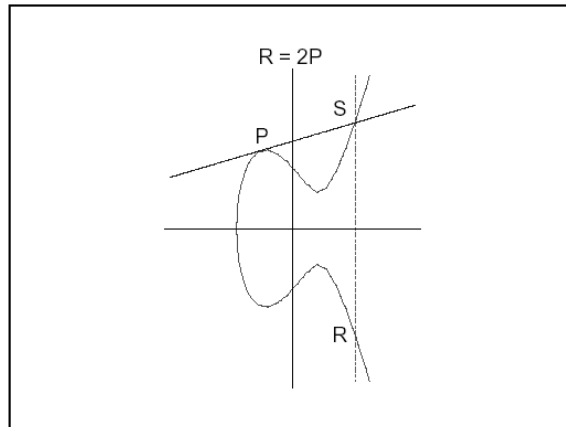


Figure 2.2: The point doubling operation ($R = 2P$) over $GF(p)$.

For a non-supersingular elliptic curve E defined over $GF(2^m)$, point addition and point doubling operations are generally computed using the algebraic formulae as follows:

- **Identity:** $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E$.
- **Negatives:** If $P = (x, y) \in E$, then $(x, y) + (x, x+y) = \mathcal{O}$. The point $(x, x+y)$ is called the negative of P , denoted as $-P$.
- **Point Addition:** Let $P = (x_1, y_1)$, $Q = (x_2, y_2) \in E$, $P \neq Q$ and $Q \neq -P$, then $P + Q = (x_3, y_3)$, where

$$x_3 = \left(\frac{y_1+y_2}{x_1+x_2}\right)^2 + \left(\frac{y_1+y_2}{x_1+x_2}\right) + x_1 + x_2 + a$$

$$y_3 = \left(\frac{y_1+y_2}{x_1+x_2}\right) \cdot (x_1 + x_3) + x_3 + y_1$$

- **Point Doubling:** If $P = Q = (x_1, y_1)$, then $2P = P + P = (x_3, y_3)$, where

$$x_3 = x_1^2 + \frac{b}{x_1^2}$$

$$y_3 = x_1^2 + \left(x_1 + \frac{y_1}{x_1}\right)x_3 + x_3$$

A major operation required by elliptic curve cryptosystems is the point scalar multiplication. The scalar multiplication operation, denoted as kP , where k is an

integer and P is a point on the elliptic curve represents the addition of k copies of point P as given by Equation 2.6.

$$kP = \underbrace{P + P + \dots + P}_{k \text{ times } P} \quad (2.6)$$

Elliptic curve cryptosystems are built over cyclic groups. Each group contains a finite number of points, n , that can be represented as scalar multiples of a generator point: iP for $i = 0, 1, \dots, n - 1$, where P is a generator of the group. The order of point P is n , which implies that $nP = \mathcal{O}$ and $iP \neq \mathcal{O}$ for $1 < i \leq n - 1$. The order of each point on the group must divide n . Consequently, a point multiplication kQ for $k > n$ can be computed as $(k \bmod n)Q$.

Projective coordinate system defines points over the projective plane as triplets (X, Y, Z) . Projective coordinate systems are used to eliminate the number of inversions [60]. For elliptic curve defined over $GF(2^m)$, many different forms of formulas may be used for point addition and doubling [61]–[64]. For the Homogeneous coordinate system, an elliptic curve point (x, y) takes the form $(x, y) = (X/Z, Y/Z)$ [62], while for the Jacobian coordinate system, a point takes the form $(x, y) = (X/Z^2, Y/Z^3)$ [63]. The Lopez-Dahab coordinate system takes the form $(x, y) = (X/Z, Y/Z^2)$ [64] and requires 14 and 5 field multiplications for point addition and point doubling respectively. This is less than the number of field multiplications required by both Homogenous and Jacobian coordinate systems. Tables

Table 2.1: The Homogeneous projective coordinate system.

Addition	Multiplications	Doubling	Multiplications
$A = X_1Z_2$	$1M$	$A = X_1Z_1$	$1M$
$B = X_2Z_1$	$1M$	$B = bZ_1^4 + X_1^4$	$1M$
$C = A + B$		$C = AX_1^4$	$1M$
$D = Y_1Z_2$	$1M$	$D = Y_1Z_1$	$1M$
$E = Y_2Z_1$	$1M$	$E = X_1^2 + D + A$	
$F = D + E$		$Z_3 = A^3$	$1M$
$G = C + F$		$X_3 = AB$	$1M$
$H = Z_1Z_2$	$1M$	$Y_3 = C + BE$	$1M$
$I = C^3 + aHC^2 + HFG$	$5M$		
$X_3 = CI$	$1M$		
$Z_3 = HC^3$	$1M$		
$Y_3 = GI + C^2[FX_1 + CY_1]$	$4M$		
Total	$16M$		$7M$

2.1–2.3 show the different formulae for point operations and the required number of field multiplications for different coordinate systems. The Mixed coordinate system adds two points where one is given in some coordinate system while the other in another coordinate system. The coordinate system of the resulting point, may be in a third coordinate system [61].

2.4 Scalar Multiplication

Scalar multiplication is the basic operation for ECC process. Scalar multiplication in the group of points of an elliptic curve is the analogous of exponentiation in

Table 2.2: The Jacobian projective coordinate system.

Addition	Multiplications	Doubling	Multiplications
$A = X_1 Z_2^2$	$1M$	$Z_3 = X_1 Z_1^2$	$1M$
$B = X_2 Z_1^2$	$1M$	$A = b Z_1^2$	$1M$
$C = A + B$		$B = X_1 + A$	
$D = Y_1 Z_2^3$	$2M$	$X_3 = B^4$	
$E = Y_2 Z_1^3$	$2M$	$C = Z_1 Y_1$	$1M$
$F = D + E$		$D = Z_3 + X_1^2 + C$	
$G = Z_1 C$	$1M$	$E = D X_3$	$1M$
$H = F X_2 + G Y_2$	$2M$	$Y_3 = X_1^4 Z_3 + E$	$1M$
$Z_3 = G Z_2$	$1M$		
$I = F + Z_3$			
$X_3 = a Z_3^2 + I F + C^3$	$3M$		
Total	$15M$		$5M$

the multiplicative group of integers modulo a fixed integer m . Computing kP can be performed using a straightforward double-and-add approach based on the binary representation of $k = (k_{m-1}, \dots, k_0)$ where k_{m-1} is the most significant bit of k . Other scalar multiplication methods have been proposed in the literature. A good survey has been conducted by Gordon in [65]. These scalar multiplication algorithms are described in detail next chapter.

Table 2.3: The Lopez-Dahab projective coordinate system.

Addition	Multiplications	Doubling	Multiplications
$A_0 = Y_1^2 Z_1^2$	$1M$	$Z_3 = Z_1^2 X_1^2$	$1M$
$A_1 = Y_1 Z_2^2$	$1M$	$X_3 = X_1^4 + bZ_1^4$	$1M$
$B_0 = X_2 Z_1$	$1M$	$Y_3 = bZ_1^4 Z_3 + X_3(aZ_3 + Y_1^2 + bZ_1^4)$	$3M$
$B_1 = X_1 Z_2$	$1M$		
$C = A_0 + A_1$			
$D = B_0 + B_1$			
$E = Z_1 Z_2$	$1M$		
$F = DE$	$1M$		
$Z_3 = F^2$			
$G = D^2(F + aE^2)$	$2M$		
$H = CF$	$1M$		
$X_3 = C_2 + H + G$			
$I = D^2 B_0 E + X_3$	$2M$		
$J = D^2 A_0 + X_3$	$1M$		
$Y_3 = HI + Z_3 J$	$2M$		
Total	$14M$		$5M$

2.5 Elliptic Curve Encryption

Several approaches were proposed to employ elliptic curves for encryption/decryption. These include elliptic curve analogs of most popular public key protocols such as elliptic curve Diffie-Hellman and elliptic curve ElGamal [59].

2.5.1 Elliptic Curve Diffie-Hellman Protocol

In Elliptic Curve Diffie-Hellman Protocol, the base point P and the elliptic curve equation are public. User's A private and public keys are k_A and $P_A = k_A P$ respectively. User's B , on the other hand, private and public keys are k_B and $P_B = k_B P$ respectively. The message to be encrypted is embedded into the x -coordinate of a point on the elliptic curve ($P_m = (x_m, y_m)$)[59]. The shared secret key S between two parties A and B is easily calculated by

$$S = k_A(k_B P) = k_B(k_A P)$$

Whenever one of the users need to send a message to the other party, he needs to add the shared secret key to the message to produce the ciphertext point P_C given by

$$P_C = P_m + S$$

To decrypt the ciphertext point, the secret key is subtracted from the ciphertext point to give the plaintext point P_m given by

$$P_m = P_c - S$$

2.5.2 Elliptic Curve ElGamal Protocol

In elliptic curve ElGamal protocol, for some user to encrypt and send the message point P_m to user A , he chooses a random integer “ l ” and generates the ciphertext which consists of the following pair of points:

$$C_m = (lP, P_m + lP_A)$$

The ciphertext pair of points uses A ’s public key, where only user A can decrypt the plaintext using his private key. To decrypt the ciphertext C_m , the first point in the pair of C_m , lP , is multiplied by A ’s private key to get the point: $k_A(lP)$. This point is subtracted from the second point of C_m to produce the plaintext point P_m .

The complete decryption operations are:

$$P_m = (P_m + lP_A) - k_A(lP) = P_m + l(k_AP) - k_A(lP)$$

2.6 Elliptic Curve Discrete Logarithm Problem

The Elliptic Curve Discrete Logarithm Problem (ECDLP) is defined as follows:

Given a known elliptic curve and two known points on the curve P and Q , the ECDLP is to find an integer $0 \leq k \leq m - 1$, such that $Q = kP$ if such a number exists.

P is called the base point while k is known as the elliptic curve discrete logarithm of Q with respect to P (i.e., $k = \log_P(Q)$). The discrete logarithm problem is an intractable problem if the parameters are carefully chosen. The most efficient algorithm known to date for computing an elliptic curve discrete logarithm is the Pollard- ρ algorithm [66].

The Pollard- ρ algorithm requires an average of $O(\sqrt{n})$, where n is the number of points on the elliptic curve, to compute an elliptic curve discrete logarithm even with its parallelized version given by Gallant *et. al.* [67]. Hence, the security of elliptic curve cryptosystems is based on the intractability of ECDLP.

It is important to realize that well-chosen curves achieve the required degree of security. Other curves may exhibit structures that facilitate cryptanalysis, e.g., curves defined over composite fields of characteristic two [50].

2.7 Summary

In this chapter a brief introduction to $GF(2^m)$ finite field arithmetic has been provided. Elements in $GF(2^m)$ are mainly represented using (1) normal basis, or (2) polynomial basis. Normal basis is chosen for the implementations described in this dissertation since field operations in normal basis mainly consist of rotation, shifting and exclusive-ORing which can be efficiently implemented in hardware.

Elliptic curve point operations have also been defined including (1) point addi-

tion, and (2) point doubling. The addition of k copies of a point P is called scalar multiplication and is denoted as kP . Scalar multiplication is the basic operation in ECC encryption/decryption process and therefore efficient scalar multiplication algorithms are highly required.

Several projective coordinate systems have been proposed to reduce the number of inversions in scalar multiplication to only one single inversion. Lopez-Dahab projective coordinate system requires less number of field multiplications as compared to other existing projective coordinate systems. Accordingly, Lopez-Dahab projective coordinate system has been selected for the the implementations presented in this dissertation.

Being the core of elliptic curve cryptosystems security, the intractability of the elliptic curve discrete logarithm problem has been also discussed in this chapter.

Chapter 3

Scalar Multiplication Algorithms

Most scalar multiplication algorithms are variants of similar algorithms employed for exponentiation. This chapter describes some of the most popular scalar multiplication algorithms [3,17,65,68–71]. These algorithms can be categorized into two main categories: (1) fixed space-time algorithms, and (2) flexible space-time algorithms.

3.1 Fixed Space-Time Scalar Multiplication Algorithms

This category contains scalar multiplication algorithms that require fixed number of point additions and doubles. These algorithms also require fixed space complexity in terms of required storage number of points.

3.1.1 Double-and-Add Scalar Multiplication Algorithm

The double-and-add scalar multiplication algorithm, so called the binary algorithm, is the easiest straightforward scalar multiplication algorithm. It inspects the bits of the scalar multiplier k , if the inspected bit $k_i = 0$, only point doubling is performed. If, however, the inspected bit $k_i = 1$, both point doubling and addition are performed. Algorithms 3.1 and 3.2 show the most-to-least and the least-to-most versions of the double-and-add scalar multiplication algorithm respectively.

In Algorithm 3.1, point doubling is always performed in Step 2.1, while point addition is performed in Step 2.2 only if $k_i = 1$. Similarly, in Algorithm 3.2, point addition is performed in Step 2.1 only if $k_i = 1$, while point doubling is always performed in Step 2.2.

The double-and-add scalar multiplication algorithm requires, m point doubles and an average of $\frac{m}{2}$ point additions. This algorithm also requires the storage of two points, P and Q .

Algorithm 3.1 Double-and-add scalar multiplication algorithm (most-to-least).

Inputs: P, k

Output: kP

Initialization:

1. $Q = P$

Scalar Multiplication:

2. for $i = m - 2$ down to 0 do

 2.1. $Q = 2Q$

 2.2. if $k_i = 1$ then $Q = Q + P$

 end for

3. return(Q)

Algorithm 3.2 Double-and-add scalar multiplication algorithm (least-to-most).

Inputs: P, k

Output: kP

Initialization:

1. $Q = \mathcal{O}, R = P$

Scalar Multiplication:

2. for $i = 0$ to $m - 1$ do

2.1. if $k_i = 1$ then $Q = Q + R$

2.2. $R = 2R$

end for

3. return(Q)

3.1.2 Addition-Subtraction Scalar Multiplication Algorithm

Instead of performing only point doubles and additions as in the double-and-add algorithm, the addition-subtraction scalar multiplication algorithm uses point subtractions in addition to point doubles and additions operations (Algorithm 3.3). Non-Adjacent Form (NAF) representation described in [71] is used in Algorithm 3.3. Using NAF representation, a multiplier $k = \sum_{i=0}^{m-1} k_i 2^i$ is uniquely recoded as $k = \sum_{i=0}^m k'_i 2^i$ with $k'_i \in [-1, 1]$, where the recoded representation does not contain contiguous nonzero digits. The NAF representation of an m -bit scalar multiplier k is at most $(m + 1)$ digits long and its average number of nonzero digits is $\frac{m}{3}$. Thus, the addition-subtraction algorithm requires m point doubles and an average of $\frac{m}{3}$ point additions.

Algorithm 3.3 shows the most-to-least version of the addition-subtraction algorithm. In Algorithm 3.3, the NAF recoding step is performed in Steps 1-1.4. Scalar multiplications of Algorithm 3.3 actually starts in Step 2. The number of iterations

performed by Algorithm 3.3 is $(m + 1)$ iterations. In Algorithm 3.3, point doubling of the accumulated point is always performed in Step 2.1, while point addition is performed in Step 2.2.1 only if the value of recoded digit $k'_i = 1$. While point subtraction is performed in Step 2.3.1 only if $k'_i = -1$.

The addition-subtraction algorithm requires only the storage of two points, P and Q .

Algorithm 3.3 Addition-subtraction scalar multiplication algorithm.

Inputs: P, k
Output: kP
Initialization:
 $Q = \mathcal{O}$
Recoding of k : $k = \sum_{i=0}^m k'_i 2^i$, $k'_i \in [-1, 1]$
1. for $i = 0$ to m do
 1.1. if $k \bmod 2 = 1$ then
 1.1.1 $k'_i = 2 - (k \bmod 2^2)$
 1.2. else
 1.2.1 $k'_i = 0$
 1.3 $k = k - k'_i$
 1.4 $k = k/2$
 end for
Scalar Multiplication:
2. for $i = m$ down to 0 do
 2.1. $Q = 2Q$
 2.2. if $k'_i = 1$ then
 2.2.1. $Q = Q + P$
 2.3. if $k'_i = -1$ then
 2.3.1. $Q = Q - P$
 end for
3. return(Q)

3.1.3 Montgomery Scalar Multiplication Algorithm

The Montgomery algorithm [72] is based on the observation that the x -coordinate of the sum of two points P_1 and P_2 , whose difference is known to be P ($P_2 - P_1 = P$), can be computed using the x -coordinates of the points P , P_1 , and P_2 . Consequently, the y -coordinate of the point P_1 can be recovered using the x -coordinates of P , P_1 , and P_2 together with the y -coordinate of P . This requires two inversions in each iteration if affine coordinate system is used as illustrated in Algorithm 3.4.

Algorithm 3.4 Montgomery Scalar Multiplication Algorithm.

Inputs: $P = (x, y)$, k

Output: kP

Initialization:

1. $x_1 = x$, $x_2 = x^2 + \frac{b}{x^2}$.

Scalar Multiplication:

2. for $i = m - 2$ down to 0 do

2.1. $t = \frac{x_1}{x_1 + x_2}$

2.2. if $k_i = 1$ then

2.2.1 $x_1 = x + t^2 + t$, $x_2 = x_2^2 + \frac{b}{x_2^2}$

2.3. else

2.3.1 $x_1 = x_1^2 + \frac{b}{x_1^2}$, $x_2 = x + t^2 + t$

end for

3. $r_1 = x_1 + x$, $r_2 = x_2 + x$.

4. $y_1 = \frac{r_1(r_1 r_2 + x^2 + y)}{x + y}$

5. return((x_1, y_1) .)

Alternatively, Lopez and Dahab [16] used projective coordinate to reduce the number of inversions to only one single inversion at the end. More details on the improved Montgomery algorithm is explained in [16].

3.2 Flexible Space-Time Scalar Multiplication Algorithms

This category contains flexible scalar multiplication algorithms that can be customized according to the user need in terms of space and time complexities.

3.2.1 w -ary Scalar Multiplication Algorithm

The double-and-add algorithm is a special case of the w -ary scalar multiplication algorithm. The w -ary algorithm processes w bits of the scalar multiplier k in each iteration instead of only a single bit as in the double-and-add algorithm. The w -ary algorithm requires recoding the scalar multiplier and precomputing some points before starting scalar multiplication.

Algorithm 3.5 shows the most-to-least version of the w -ary algorithm. Steps 1-1.3 show the process of recoding the multiplier k using radix digits. The recoded scalar multiplier has $\lceil \frac{m}{w} \rceil$ with each digit in the range $[0, 2^w)$: $k = \sum_{i=0}^{\lceil \frac{m}{w} \rceil - 1} k'_i 2^{wi}$. The points iP for $i \in [2, 2^w)$ are precomputed (Steps 2-2.2), and stored to be used later as needed. The precomputed points require approximately 2^{w-1} point doubles and 2^{w-1} point additions.

The scalar multiplication process starts at Step 3. The number of iterations performed is $\lceil \frac{m}{w} \rceil$. In Algorithm 3.5, w point doubles are always performed in Step 3.1, while point addition of is performed in Step 3.2 only if $k'_i \neq 0$.

The scalar multiplication phase requires m point doubles and approximately an average of $\frac{m}{w}$ point additions. The algorithm also requires the storage of approximately 2^w points.

Algorithm 3.5 w -ary scalar multiplication algorithm.

Inputs: P, k
Output: kP
Initialization:
 $Q = \mathcal{O}, P_1 = P$
Recoding of k : $k = \sum_{i=0}^{\lceil \frac{m}{w} \rceil - 1} k'_i 2^{wi}, k'_i \in [0, 2^w)$
1. for $i = 0$ to $\lceil \frac{m}{w} \rceil - 1$ do
 1.1. $k'_i = k \bmod 2^w$
 1.2. $k = k - k'_i$
 1.3. $k = k/2^w$
end for
Precomputations: $P_i = iP, i \in [0, 2^w)$
2. for $i = 1$ to $2^{w-1} - 1$ do
 2.1. $P_{2i} = 2P_i$
 2.2. $P_{2i+1} = P_{2i} + P$
end for
Scalar Multiplication:
3. for $i = \lceil \frac{m}{w} \rceil - 1$ down to 0 do
 3.1. $Q = 2^w Q$
 3.2. if $k'_i \neq 0$ then
 3.2.1. $Q = Q + P_{k'_i}$
 end for
4. return(Q)

3.2.2 w -ary Addition-subtraction Scalar Multiplication Algorithm

The addition-subtraction algorithm is a special case of the w -ary addition-subtraction scalar multiplication algorithm. The w -ary addition-subtraction algorithm processes

w bits of the scalar multiplier k in each iteration instead of only a single bit as in the addition-subtraction algorithm. The w -ary algorithm requires recoding of the scalar multiplier and precomputing of some points before starting the scalar multiplication process.

Algorithm 3.6 w -ary addition-subtraction scalar multiplication algorithm.

Inputs: P, k

Output: kP

Initialization:

$Q = \mathcal{O}, P_1 = P$

Recoding of k : $k = \sum_{i=0}^{\lceil \frac{m+1}{w} \rceil - 1} k'_i 2^{wi}, k'_i \in [-2^{w-1}, 2^{w-1})$

1. for $i = 0$ to $\lceil \frac{m+1}{w} \rceil - 1$ do

1.1. $k'_i = k \bmod 2^w$

1.2. if $k'_i \geq 2^{w-1}$ then

1.2.1 $k'_i = -(2^w - k'_i)$

1.3. $k = k - k'_i$

1.4. $k = k / 2^w$

end for

Precomputations: $P_i = iP, i \in [1, 2^{w-1}]$

2. for $i = 1$ to $2^{w-2} - 1$ do

2.1. $P_{2i} = 2P_i$

2.2. $P_{2i+1} = P_{2i} + P$

end for

3. $P_{2^{w-1}} = 2P_{2^{w-2}}$

Scalar Multiplication:

4. for $i = \lceil \frac{m+1}{w} \rceil - 1$ down to 0 do

4.1. $Q = 2^w Q$

4.2. if $k'_i > 0$ then

4.2.1. $Q = Q + P_{k'_i}$

4.3. if $k'_i < 0$ then

4.3.1. $Q = Q - P_{|k'_i|}$

end for

5. return(Q)

Algorithm 3.6 shows the most-to-least version of the w -ary addition-subtraction algorithm. Steps 1-1.4 show the recoding of the multiplier k in radix 2^w with $\lceil \frac{m+1}{w} \rceil$

digits in the range $[-2^{w-1}, 2^{w-1})$: $k \sum_{i=0}^{\lceil \frac{m+1}{w} \rceil - 1} k'_i 2^{wi}$. Precomputations are performed in Steps 2-3. The precomputed points are of the values iP for $i \in [2, 2^{w-1}]$ and require approximately 2^{w-2} point doubles and 2^{w-2} point additions.

Scalar multiplications of Algorithm 3.6 actually starts in Step 4. The number of iterations performed by Algorithm 3.6 is $\lceil \frac{m+1}{w} \rceil - 1$ iterations. In Algorithm 3.6, w point doubles of the accumulated point are always performed in Step 4.1, while point addition of the point $P_{|k'_i|}$ and the accumulated point is performed in Step 4.2.1 only if $k'_i > 0$. If $k'_i < 0$, the point $P_{|k'_i|}$ is subtracted from the accumulated point in Step 4.3.1.

The w -ary algorithm requires in scalar multiplication phase m point doubles and an average of $\frac{m}{w}$ point additions. The algorithm also requires the storage of approximately 2^{w-1} points.

3.2.3 Width- w Addition-Subtraction Scalar Multiplication

Algorithm

The width- w addition-subtraction scalar multiplication algorithm is an extension of the addition-subtraction algorithm. It uses w -NAF recoding to recode the scalar multiplier k as follows: $k = \sum_{i=0}^l k'_i 2^i$ where $k'_i \in (-2^{w-1}, 2^{w-1})$ and k'_i is odd. The width- w addition-subtraction algorithm requires also precomputation in addition to recoding before performing scalar multiplication.

Algorithm 3.7 Width- w Addition-Subtraction Scalar Multiplication Algorithm.

Inputs: P, k

Output: kP

Initialization:

$Q = \mathcal{O}$

Recoding of k : $k = \sum_{i=0}^m k'_i 2^i$, $k'_i \in (-2^{w-1}, 2^{w-1})$

1. for $i = 0$ to m do
 - 1.1. if $k \bmod 2 = 1$ then
 - 1.1.1. $k'_i = k \bmod 2^w$
 - 1.1.2. if $k'_i \geq 2^{w-1}$ then
 - 1.1.2.1 $k'_i = -(2^w - k'_i)$
 - 1.2. else
 - 1.2.1. $k'_i = 0$
 - 1.3. $k = k - k'_i$
 - 1.4. $k = k/2$
- end for

Precomputations:

2. $P_0 = P$
 3. $T = 2P$
 4. for $i = 1$ to $2^{w-2} - 1$ do
 - 4.1 $P_i = P_{i-1} + T$
- end for

Scalar Multiplication:

5. for $i = m$ down to 0 do
 - 5.1. $Q = 2Q$
 - 5.2. if $k'_i > 0$ then
 - 5.2.1. $Q = Q + P_{\lfloor k'_i/2 \rfloor}$
 - 5.3. if $k'_i < 0$ then
 - 5.3.1. $Q = Q - P_{\lfloor |k'_i|/2 \rfloor}$
 - end for
 6. return(Q)
-

Algorithm 3.7 shows the most-to-least version of the width- w addition-subtraction algorithm. In Algorithm 3.7, the w -NAF recoding step is performed in Steps 1-1.4. Precomputation of the points iP for odd values of i in the range $[3, 2^{w-1})$ are computed in Steps 2-4.1. The precomputation phase of Algorithm 3.7 requires one point double and $2^{w-2} - 1$ point additions.

Scalar multiplications of Algorithm 3.7 actually starts in Step 5. The number of iterations performed by Algorithm 3.7 is $m + 1$ iterations. In Algorithm 3.7, point doubling of the accumulated point is always performed in Step 5.1. Point addition is performed between the accumulated point and the point $P_{\lfloor k'_i/2 \rfloor}$ in Step 5.2.1 only if the value of recoded digit $k'_i > 0$. While the point $P_{\lfloor k'_i/2 \rfloor}$ is subtracted from the accumulated point in Step 5.3.1 only if $k'_i < 0$. Scalar multiplications require m point doubles and on the average $\frac{m}{w+1}$ point additions/subtractions. The algorithm also requires the storage of approximately 2^{w-2} points.

3.2.4 Signed BGMW Scalar Multiplication Algorithm

The signed BGMW scalar multiplication algorithm is based on the BGMW scalar multiplication algorithm [73]. It recodes the scalar multiplier k using signed digit representation as: $k = \sum_{i=0}^{\lceil \frac{m+1}{w} \rceil - 1} k'_i 2^{wi}$ with $k'_i \in (-2^{w-1}, 2^{w-1})$.

Algorithm 3.8 shows the signed BGMW scalar multiplication algorithm. In Algorithm 3.8, recoding of the scalar multiplier k is performed in Steps 1-1.4. The precomputation steps (Steps 2-3.1), require precomputing the points $2^{wi}P$ for

Algorithm 3.8 Signed BGMW Scalar Multiplication Algorithm.

Inputs: P, k

Output: kP

Initialization:

$A = \mathcal{O}, B = \mathcal{O}$

Recoding of k : $k = \sum_{i=0}^{\lceil \frac{m+1}{w} \rceil - 1} k'_i 2^{wi}, k'_i \in [-2^{w-1}, 2^{w-1})$

1. for $i = 0$ to $\lceil \frac{m+1}{w} \rceil - 1$ do

1.1. $k'_i = k \bmod 2^w$

1.2. if $k'_i \geq 2^{w-1}$ then

1.2.1 $k'_i = -(2^w - k'_i)$

1.3. $k = k - k'_i$

1.4. $k = k/2^w$

end for

Precomputation: $P_i = 2^{wi}P, \forall i = 0, 1, \dots, (\lceil \frac{m+1}{w} \rceil - 1)$

2. $P_0 = P$

3. for $i = 1$ to $\lceil \frac{m+1}{w} \rceil - 1$ do

3.1. $P_i = 2^{wi}P_{i-1}$

end for

Scalar Multiplication:

4. for $j = 2^{w-1}$ down to 1 do

4.1. for each i for which $k'_i = j$ do

4.1.1 $B = B + P_i$

4.2. for each i for which $k'_i = -j$ do

4.2.1 $B = B - P_i$

4.3. $A = A + B$

5. return(A)

$$i = 1, 2, \dots, (\lceil \frac{m+1}{w} \rceil - 1).$$

In the scalar multiplication phase, point addition or subtraction is performed between the accumulated point and the point $2^{wi}P$ depending on the value of k'_i . From the iteration at which the point $2^{wi}P$ is added or subtracted till the last loop iteration, the accumulated point is added to itself k'_i times; therefore, the accumulated point incorporates the point $k'_i(2^{wi}P)$ in its result (Steps 4-4.3).

In Algorithm 3.8, scalar multiplication requires $2^{w-1} + \frac{m}{w}$ point additions/subtractions and no point doubling is required. The algorithm also requires the storage of $\lceil \frac{m}{w} \rceil$ points.

3.2.5 Lim-Lee Scalar Multiplication Algorithm

The Lim-Lee scalar multiplication algorithm, so called the comb scalar multiplication algorithm, is proposed by Lim and Lee in [74]. Algorithm 3.9 shows the most-to-least version of the Lim-Lee scalar multiplication algorithm.

In this algorithm, the scalar multiplier k is arranged into h blocks, each of length $a = \lceil \frac{m}{h} \rceil$. Furthermore, each block is subdivided into v blocks of size $b = \lceil \frac{a}{v} \rceil$. Thus, the scalar multiplier k can be written as:

$$k = \sum_{r=0}^{h-1} \sum_{s=0}^{v-1} \sum_{t=0}^{b-1} k_{vbr+bs+t} 2^{vbr+bs+t}$$

The scalar multiplication expression is given as:

$$kP = \sum_{t=0}^{b-1} 2^t \left(\sum_{s=0}^{v-1} G[s][I_{s,t}] \right)$$

Algorithm 3.9 Lim-Lee Scalar Multiplication Algorithm.

Inputs:

P, k

h - number of blocks of k = the number of rows in the precomputation array

a - length of the blocks

v - number of sub-blocks to which each block is divided

b - size of each sub-block

Output: kP

Initialization:

$Q = \mathcal{O}$

Precomputation:

1. for $i = 0$ to $h - 1$ do

 1.1. $P_i = 2^{ai}P$

 end for

Precomputation array:

2. for $u = 1$ to $2^h - 1$ do

 2.1. $G[0][u] = \sum_{s=0}^{h-1} u_s P_s$, where $u = \sum_{s=0}^{h-1} u_s 2^s$, $u_s \in [0, 1]$

 2.2. for $s = 1$ to $v - 1$ do

 2.2.1. $G[s][u] = 2^{sb} G[0][u]$

 end for

 end for

Scalar Multiplication:

3. for $t = b - 1$ down to 0 do

 3.1. $Q = 2Q$

 3.2. for $s = v - 1$ down to 0 do

 3.2.1 $I_{s,t} = \sum_{i=0}^{h-1} k_{at+bs+t} 2^i$

 3.2.2 if $I_{s,t} \neq 0$ then

 3.2.2.1. $Q = Q + G[s][I_{s,t}]$

 end for

 end for

4. return(Q)

where the precomputation array $G[s][u]$ for $0 \leq s < v$, $0 \leq u < 2^h$, and $u = (u_{h-1} \dots u_0)_2$, is defined by the following equations:

$$\begin{aligned} G[0][u] &= \sum_{r=0}^{h-1} u_r 2^{rvb} P, \\ G[s][u] &= 2^{sb} G[0][u], \end{aligned}$$

and the number $I_{s,t}$, for $0 \leq s < v - 1$ and $0 \leq t < b$ is defined by:

$$I_{s,t} = \sum_{r=0}^{h-1} k_{vbr+bs+t} 2^r,$$

Algorithm 3.9 requires $b - 1$ point doubles and an average of a point additions. This algorithm also requires the storage of $v(2^h - 1)$ points. More details on Lim-Lee algorithm is explained in [74].

Table 3.1 summarizes the average time complexity and storage requirement of scalar multiplication algorithms. The average time complexity is specified in terms of point additions and point doubles without including precomputations which can be performed off-line. While the storage requirements are specified in terms of the number of points that needs to be stored.

3.3 Summary

In this chapter, scalar multiplication algorithms have been surveyed. These algorithms can be categorized into two main categories: (1) fixed space-time algorithms,

Table 3.1: Complexity of scalar multiplication algorithms

Algorithm	# doubles	# additions	# Stored points
double-and-add	m	$\frac{m}{2}$	2
addition-subtraction	m	$\frac{m}{3}$	2
w -ary	m	$\frac{m}{w}$	2^w
w -ary addition-subtraction	m	$\frac{m}{w}$	2^{w-1}
width- w addition-subtraction	m	$\frac{m}{w+1}$	2^{w-2}
Signed BGMW	0	$2^{w-1} + \frac{m}{w}$	$\frac{m}{w}$
Lim-Lee ($m = ah$, $a = vb$)	$b - 1$	a	$v(2^h - 1)$

and (2) flexible space-time algorithms. Most of these algorithms use scalar multiplier recoding and precomputed points to speedup the required time for scalar multiplication.

The fixed space-time algorithms category contains scalar multiplication algorithms that require fixed number of point additions and doubles. These algorithms also require fixed storage requirements. These include (1) the double-and-add algorithm, (2) the addition-subtraction algorithm, and (3) the Montgomery algorithm.

The flexible space-time algorithms category, on the other hand, contains flexible scalar multiplication algorithms that can be customized according to user needs in terms of space and time. These include (1) the w -ary algorithm, (2) the w -ary addition-subtraction algorithm, (3) the width- w addition-subtraction algorithm, (4) the Signed BGMW algorithm, and (5) the Lim-Lee algorithm.

For the implementations presented in this dissertation, the double-and-add al-

gorithm is selected for one implementation and new algorithms are proposed for the other implementations which use scalar multiplier recoding and precomputed points.

Chapter 4

Normal Basis $GF(2^m)$ Field

Arithmetic

Efficient computations in finite fields and their architectures are important in many applications, including coding theory, and public-key cryptosystems (e.g., elliptic curve cryptosystems (ECC) [1]). Although all finite fields of the same cardinality are isomorphic, their arithmetic efficiency depends greatly on the choice of the basis used for field element representation. The most commonly used bases are the polynomial basis (PB) and the normal basis (NB)[60][75]. The normal basis [57] is more suitable for hardware implementations than the polynomial basis since operations in normal basis representation are mainly comprised of rotation, shifting and exclusive-ORing which can be efficiently implemented in hardware. This chapter surveys $GF(2^m)$ field multiplication and inversion algorithms. A more detailed survey with examples

is found in our work in [76].

4.1 Multiplication

In finite field arithmetic, multiplication is a more complex operation than addition and squaring. An efficient multiplier is the key for efficient finite field computations.

Finite field multipliers using normal basis can be classified into two main categories:

(1) λ -matrix based multipliers and (2) Conversion based multipliers.

4.1.1 λ -Matrix Based Multipliers

Massey and Omura [77] proposed an efficient normal basis bit-serial multiplier over $GF(2^m)$. The Massey-Omura multiplier requires only two m -bit cyclic shift registers and combinational logic. The combinational logic consists of a set of AND and XOR logic gates (see Figure 4.1). The first implementation of the Massey-Omura multiplier was reported by Wang. *et. al.* [78]. The space complexity of the Massey-Omura multiplier is $(2m - 1)$ AND gates + $(2m - 2)$ XOR gates, while the time complexity is $T_A + (1 + \log_2(m - 1))T_X$, where T_A and T_X are the delay of one AND gate and one XOR gate respectively. One advantage of the Massey-Omura multiplier is that it can be used with both types of the optimal normal basis (Type I and Type II). The bit-parallel version of the Massey-Omura multiplier requires $(2m^2 - m)$ AND gates + $(2m^2 - 2m)$ XOR gates.

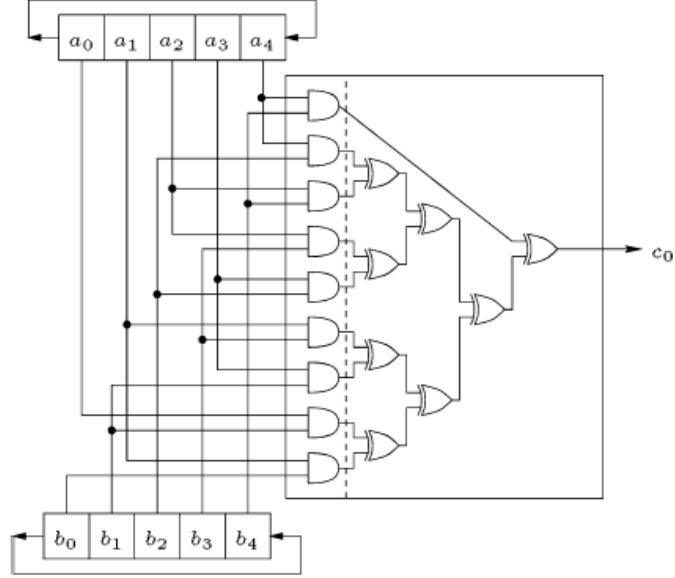


Figure 4.1: $GF(2^5)$ bit-serial Massey-Omura multiplier [77].

Hasan *et. al.* [79] proposed a modified version of the Massey-Omura parallel multiplier which works only with ONB Type I. The basic idea is to decompose the λ^{m-1} matrix as a sum of two matrices P and Q .

$$\lambda^{(m-1)} = P + Q \pmod{2}$$

where the (i, j) entry of P is defined as follows:

$$p_{i, j} = \begin{cases} 1, & \text{if } i = (\frac{m}{2} + j); \\ 0 & \text{otherwise.} \end{cases}$$

The product c_{m-1-k} can be obtained as:

$$c_{m-1-k} = A \lambda^{(k)} B^t = A P B^t + A^{(k)} Q B^{(k)t} \triangleq \hat{c} + \hat{c}_{m-1-k} \pmod{2}$$

where $A^{(k)}$ and $B^{(k)}$ are the k -cyclic shifted vectors of A and B . Hasan *et al.* noticed that \hat{c} is independent of k and is present in each c_{m-1-k} . Hence, it can be precomputed once at the beginning thus reducing the multiplier's hardware complexity. Compared to the Massey-Omura parallel multiplier, this multiplier requires only $(m^2 - 1)$ XOR gates and the same number of AND gates. However, the time complexity of this modified parallel multiplier is the same as the Massey-Omura parallel multiplier and the number of XOR gates is still $O(m^2)$.

Alternatively, Gao and Sobelman in [80] noticed that the Massey-Omura bit-serial multiplier is constructed using an AND plane and an XOR plane. Gao and Sobelman suggested to rearrange these planes into three planes: XOR-plane, AND-plane and another XOR-plane as shown in Figure 4.2. This is achieved by rearranging the multiplication equation as:

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i \lambda_{ij}^{(k)} b_j = \sum_{i=0}^{m-1} a_i \left[\sum_{j=0}^{m-1} \lambda_{ij}^{(k)} b_j \right]$$

The Gao and Sobelman multiplier [80] requires the same number of XOR gates and has the same time complexity as the Massey-Omura multiplier. The only improvement was reducing the number of AND gates to only m^2 AND gate compared to $(2m^2 - m)$ AND gates for the Massey-Omura multiplier.

Reyhani-Masoleh and Hasan [81] presented another multiplier based on the same idea of rearranging the multiplier's XOR and AND planes as in [80] but with a different formulation. The product terms are reformulated as:

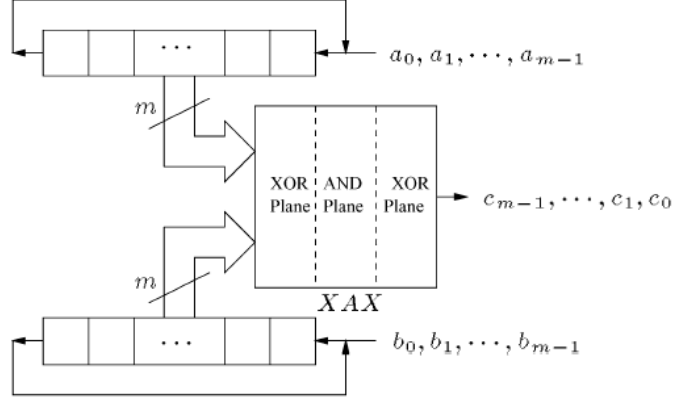


Figure 4.2: Gao and Sobelman multiplier [80].

$$c_k = a_k b_k + \sum_{(r, s) \in \Phi_k} (a_r + a_s)(b_r + b_s), \quad 0 \leq k \leq m-1.$$

where Φ_k contains the coordinates of 1's in the upper part of the λ_k matrix. The space complexity is (m^2) AND + $(3m^2 - 3m)$ XOR gates while the time complexity is $T_A + \lceil \log_2(2m-1) \rceil T_X$. This means that the Gao and Sobelman [80] multiplier is more efficient than Reyhani-Masoleh and Hasan multiplier [81].

Reyhani-Masoleh and Hasan have also reported [82] a parallel multiplier that takes advantage of the symmetry of the λ matrix and reduced redundancy in the λ matrix. This is achieved by rewriting the λ matrix as $\lambda = U + U^T + D$, where D is the diagonal matrix and U is the upper triangular matrix having zeros at diagonal entries. Thus, the multiplication product can be written as follows:

$$C = A \times U \times B^T + B \times U \times A^T + A \times D \times B^T$$

Table 4.1: The λ -based multipliers and their space and time complexities.

Multiplier	Space Complexity	Time Complexity	Type of ONB
Massey and Omura [77]	$(2m^2 - m)$ AND + $(2m^2 - 2m)$ XOR	$T_A + (1 + \log_2(m - 1))T_X$	I & II
Hasan <i>et. al.</i> [79]	$(2m^2 - m)$ AND + $(m^2 - 1)$ XOR	$T_A + (1 + \log_2(m - 1))T_X$	I
Gao and Sobelman [80]	(m^2) AND + $(2m^2 - 2m)$ XOR	$T_A + (1 + \log_2(m - 1))T_X$	I & II
Reyhani-Masoleh and Hasan [81]	(m^2) AND + $(3m^2 - 3m)$ XOR	$T_A + \lceil \log_2(2m - 1) \rceil T_X$	I & II
Reyhani-Masoleh and Hasan [82]	(m^2) AND + $(m^2 - 1)$ XOR	$T_A + (1 + \log_2(m - 1))T_X$	I & II

The U matrix is reformulated according to the value of m (even or odd). However, the proposed multiplier's space complexity is (m^2) AND + $(m^2 - 1)$ XOR, and the time complexity is $T_A + (1 + \log_2(m - 1))T_X$ which represents minor gain as compared to other multipliers.

Table 4.1 compares the λ -based multipliers space and time complexities. The space complexity is specified by the number of AND and XOR gates. While the time complexity is specified by time required by the critical path to produce the results.

4.1.2 Conversion Based Multipliers

Koc and Sunar in [83] proposed a new bit-parallel multiplier over $GF(2^m)$. The multiplier was employed to perform Type I optimal normal basis multiplication by converting the two operands into canonical basis¹. After multiplication, the result is converted back to the normal basis. The proposed technique yields a slight improvement in the number of XOR gates as compared to the Massey-Omura multiplier. The number of required XOR gates is reduced down to $(m^2 - 1)$ as compared to $(2m^2 - m)$ for the Massey-Omura multiplier, while its time complexity $T_A + (2 + \log_2(m - 1))T_X$ is slightly more than Massey-Omura's time complexity. The main advantage of this technique, however, is the opening of a new direction in multiplications based on basis conversion which was first explored by Sunar and Koc in [84].

Sunar and Koc in [84] reported a new Type II optimal normal basis multiplier. The main idea of their work is based on converting the two operands to equivalent representations in another basis, perform the multiplication in that basis and convert the product back to the normal basis. The conversion step requires only a single clock cycle since it is nothing but a permutation of the normal basis.

Using optimal normal basis Type II and assuming that $p (= 2m + 1)$ is prime, another new basis which is obtained by simple permutation of the normal basis elements was presented. Let β be $\gamma + \gamma^{-1}$, where γ is the primitive p th root of unity,

¹A basis of the form $(\alpha^{m-1}, \dots, \alpha^2, \alpha^1, 1)$, where $\alpha \in GF(2^m)$ is a root of the generating polynomial of degree m , is called a canonical basis.

the new basis can be written in the form $\gamma + \gamma^{-1}, \gamma^2 + \gamma^{-2}, \gamma^3 + \gamma^{-3}, \dots, \gamma^m + \gamma^{-m}$.

Two elements \hat{A} and $\hat{B} \in GF(2^m)$ can be represented in the new basis as:

$$\begin{aligned}\hat{A} &= \sum_{i=1}^m \hat{a}_i(\gamma^i + \gamma^{-i}) = \sum_{i=1}^m \hat{a}_i \beta_i, \text{ and} \\ \hat{B} &= \sum_{i=1}^m \hat{b}_i(\gamma^i + \gamma^{-i}) = \sum_{i=1}^m \hat{b}_i \beta_i\end{aligned}$$

The product $\hat{C} = \hat{A} \cdot \hat{B}$ is written as:

$$\hat{C} = \left(\sum_{i=1}^m \hat{a}_i(\gamma^i + \gamma^{-i}) \right) \left(\sum_{j=1}^m \hat{b}_j(\gamma^j + \gamma^{-j}) \right)$$

This product can be transformed to the following form:

$$\begin{aligned}\hat{C} &= \sum_{i=1}^m \sum_{j=1}^m \hat{a}_i \hat{b}_j (\gamma^{i-j} + \gamma^{-(i-j)}) + \sum_{i=1}^m \sum_{j=1}^m \hat{a}_i \hat{b}_j (\gamma^{i+j} + \gamma^{-(i+j)}) \\ &= \hat{C}_1 + \hat{C}_2\end{aligned}$$

The term \hat{C}_1 has the property that the exponent $(i - j)$ of γ is already within the proper range, i.e., $-m \leq (i - j) \leq m$ for all $i, j \in [1, m]$. The term \hat{C}_2 should be ensured to be in the proper range. Hence, \hat{C}_2 is computed as follows:

$$\begin{aligned}\hat{C}_2 &= \sum_{i=1}^m \sum_{j=1}^m \hat{a}_i \hat{b}_j (\gamma^{i+j} + \gamma^{-(i+j)}) \\ &= \sum_{i=1}^m \sum_{j=1}^{m-i} \hat{a}_i \hat{b}_j (\gamma^{i+j} + \gamma^{-(i+j)}) + \sum_{i=1}^m \sum_{j=m-i+1}^m \hat{a}_i \hat{b}_j (\gamma^{i+j} + \gamma^{-(i+j)}) \\ &= \hat{D}_1 + \hat{D}_2\end{aligned}$$

The exponents of the basis elements $\gamma^{i+j} + \gamma^{-(i+j)}$ in \hat{D}_1 are guaranteed to be in the proper range $1 \leq (i+j) \leq m$ for $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, m-i$. If $k = i+j$, then product $\hat{a}_i \hat{b}_j$ contributes to the basis element β_k as i and j take these values. The basis elements of \hat{D}_2 , however, are all out of range. Thus, the identity γ^{2m+1} is used to bring them to the proper range as:

$$\begin{aligned}\hat{D}_2 &= \sum_{i=1}^m \sum_{j=m-i+1}^m \hat{a}_i \hat{b}_j (\gamma^{i+j} + \gamma^{-(i+j)}) \\ &= \sum_{i=1}^m \sum_{j=m-i+1}^{m-i} \hat{a}_i \hat{b}_j (\gamma^{2m+1-(i+j)} + \gamma^{-(2m+1-(i+j))})\end{aligned}$$

Therefore, if $k = i+j > m$, β_k is replaced by β_{2m+1-k} and thus the final product can be found as:

$$\hat{C} = \hat{C}_1 + \hat{D}_1 + \hat{D}_2$$

The hardware complexity of this bit-parallel multiplier is m^2 AND gates + $\frac{3}{2}(m^2 - m)$ XOR gates and the time complexity is $T_A + (1 + \lceil \log_2 m \rceil)T_X$. This represents an improvement of about 25 percent less XOR gates compared to the Massey-Omura multiplier but with slightly more delay. A major advantage of this method, however, is the fact that there is no need to store the λ matrix to perform multiplications which requires m^3 locations if all λ matrices are stored or m^2 if only $\lambda^{(0)}$ is stored. This is a major advantage in area constrained environments since we only need to store the permutation array which requires only $m \log_2(m)$ storage locations.

Alternatively, Wu *et. al.* in [85] extended the work described in [86] and presented a new Type II multiplier. The basic idea is the same as that of Sunar and Hasan [84]. Multiplication is performed by converting the two operands into another basis which is simply a permutation of the normal basis, perform the multiplication and convert the product back to the original normal basis. The difference is only in the multiplication part. The product can be found in the new basis as:

$$\hat{c}_j = \sum_{i=1}^m \hat{a}_i (\hat{b}_{s(j+i)} + \hat{b}_{s(j-i)}), \quad j = 1, 2, \dots, m.$$

where $s(i)$ is defined as:

$$s(i) = \begin{cases} i \bmod 2m + 1, & \text{if } 0 \leq i \bmod 2m + 1 \leq m; \\ 2m + 1 - i \bmod 2m + 1, & \text{otherwise.} \end{cases}$$

The hardware complexity of this multiplier is m^2 AND gates + $(2m^2 - m)$ XOR gates which is worse than the Sunar and Koc multiplier [84] as shown in Figure 4.3. However, the time complexity is $T_A + (1 + \lceil \log_2 m \rceil)T_X$ which is the same as the Sunar and Koc multiplier [84].

Table 4.2 summarizes the conversion based multipliers and their space and time complexities. It is clear from Table 4.2 that Sunar and Koc multiplier provides the best space and time complexities.

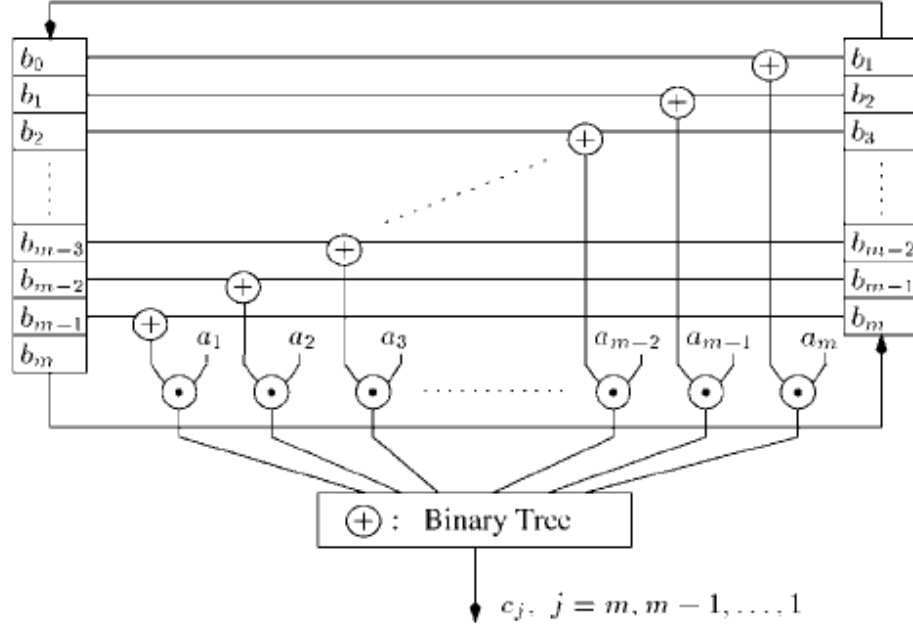


Figure 4.3: The Wu *et. al.* multiplier [85].

Table 4.2: The conversion based multipliers and their space and time complexities.

Multiplier	Space Complexity	Time Complexity	Type of ONB
Koc and Sunar [83]	$(2m^2 - m)$ AND + $(m^2 - 1)$ XOR	$T_A + (2 + \log_2(m - 1))T_X$	I
Sunar and Koc [84]	m^2 AND + $\frac{3}{2}(m^2 - m)$ XOR	$T_A + (1 + \lceil \log_2 m \rceil)T_X$	II
Wu <i>et. al.</i> [85]	m^2 AND + $(2m^2 - m)$ XOR	$T_A + (1 + \lceil \log_2 m \rceil)T_X$	II

4.2 Inversion

Inversion using normal basis consists of multiplications and cyclic shifts. Since cyclic shifts require almost negligible time, the number of multiplications is the key parameter for efficient inversion. In the following subsections we survey existing algorithms for inversion over $GF(2^m)$ using normal basis. These inversion algorithms can be classified into three main categories: (1) Standard, (2) Exponent Decomposing and (3) Exponent Grouping inversion algorithms.

4.2.1 Standard Inversion Algorithm

This category contains only one algorithm, which is the first proposed normal basis inversion algorithm over $GF(2^m)$ by Wang *et. al.* [78]. The basic idea is derived from Fermat's Little Theorem where the inverse of $a \in GF(2^m)$ is given by $a^{-1} = a^{2^m-2}$. Since $2^m - 2 = \sum_{i=1}^{m-1} 2^i$, we can express a^{-1} as:

$$a^{-1} = (a^2).(a^{2^2})...(a^{2^{m-1}}) = a^{2^m-2}$$

Algorithm 4.1 Wang's *et. al.* inversion algorithm.

Inputs: a

Output: a^{-1}

1. $B = a^2$, $C = 1$ and $k = 0$
 2. $D = B \times C$ and $k = k + 1$
 3. if $k = m - 1$, $a^{-1} = D$ Stop
 4. if $k < m - 1$, $B = B^2$ and $C = D$
 5. Go back to 2
-

This only requires multiplication and cyclic shift operations. The algorithm procedure for computing a^{-1} as suggested by Wang *et. al.* [78] is shown in Algorithm 4.1. It requires $(m-2)$ multiplications + $(m-1)$ cyclic shifts. The advantage of this method is its simplicity while its disadvantage is the large $O(m)$ number of multiplications.

4.2.2 Exponent Decomposing Inversion Algorithms

Since the number of multiplications is the dominant factor in determining the computation time of the inversion operation, several algorithms attempted to improve the inversion speed by decomposing the exponent to reduce the required number of multiplications and replace it with squaring operations which are much simpler compared to multiplication.

In 1988 Itoh and Tsujii proposed a $GF(2^m)$ inversion algorithm derived from Fermat's Little Theorem using normal basis [87]. The basic idea used was to decompose the exponent $m-1$ as follows:

$$a^{-1} = a^{2^m-2} = (a^{2^{m-1}-1})^2$$

The exponent $2^{m-1} - 1$ is further decomposed as follows:

1. If m is odd, then

$$(2^{m-1} - 1) = (2^{\frac{m-1}{2}} - 1)(2^{\frac{m-1}{2}} + 1), \text{ and}$$

$$a^{2^{m-1}} = (a^{2^{\frac{m-1}{2}}-1})^{2^{\frac{m-1}{2}}+1}$$

2. If m is even, then

$$2^{m-1} - 1 = 2(2^{m-2} - 1) + 1 = 2(2^{\frac{m-2}{2}} - 1)(2^{\frac{m-2}{2}} + 1) + 1, \text{ and}$$

$$a^{2^{m-1}} = a^{2(2^{\frac{m-2}{2}}-1)(2^{\frac{m-2}{2}}+1)+1}$$

The proposed algorithm by Itoh and Tsujii [87] is shown in Algorithm 4.2 and it requires $\log_2(m-1) + v(m-1) - 1$ multiplications, where $v(x)$ is the number of 1's in the binary representation of x .

Algorithm 4.2 Itoh-Tsujii inversion algorithm.

Inputs: a
Output: $l = a^{-1}$

1. set $s \leftarrow \lfloor \log_2(m-1) \rfloor - 1$
2. set $p \leftarrow a$
3. for $i = s$ down to 0 do
 - 3.1. set $r \leftarrow$ shift $m-1$ to right by s bit(s)
 - 3.2. set $q \leftarrow p$
 - 3.3. rotate q to left by $\lfloor r/2 \rfloor$ bit(s)
 - 3.4. set $t \leftarrow p \times q$
 - 3.5. if least bit of $r = 1$
 - 3.5.1 rotate t to left by 1 bit
 - 3.5.2 $p \leftarrow t \times a$
 - 3.5. else
 - 3.5.3 $p \leftarrow t$
 - 3.6. $s \leftarrow s - 1$
4. rotate p to left by 1 bit
5. set $l \leftarrow p$
6. return l

Feng [88] has also proposed an inversion algorithm which requires the same time complexity as the Itoh and Tsujii inversion algorithm, i.e. $O(\log_2(m))$. The inversion algorithm was also derived from Fermat's Little Theorem and is also based on exponent decomposition as the Itoh and Tsujii inversion algorithm [87]. Feng defined $m - 1$ as follows:

Let $m_q m_{q-1} m_1 m_0$ be the binary representation of $(m - 1)$, where $m_q = 1$ and m_i is 0 or 1 for $i = 0$ to $q - 1$, i.e. $m - 1 = m_q 2^q + m_{q-1} 2^{q-1} + \dots + m_1 2^1 + m_0 2^0$. The inverse a^{-1} can be computed using Algorithm 4.3.

Algorithm 4.3 Feng's inversion algorithm.

Inputs: a

Output: a^{-1}

1. $b = a$
 2. for $i = q$ to 1 do
 - 2.1. if $m_i = 1$, then $b = b^{2^{-2^i}}$
 - 2.2. $b = b \times b^{2^{-2^{i-1}}}$
 - 2.3. if $m_{i-1} = 1$, then $b = b \times a$
 3. $a^{-1} = (b)^{2^{m-m_0}}$, Stop
-

Only steps 2.2 and 2.3 in Algorithm 4.3 require multiplications. Step 2.1 and 3 need only cyclic shifts. Thus, the algorithm has $(q + p)$ multiplications where $q = \lfloor \log_2(m) \rfloor$ and, $p = \#1s$ in the binary expression of $(m - 1)$. Accordingly, the proposed algorithm has the same complexity as the Itoh and Tsujii inversion algorithm [87]. The difference is only that the Itoh and Tsujii algorithm performs squaring during each iteration, while Feng's algorithm computes the square roots each iteration and squares at the end by 2^{m-m_0} .

Takagi *et. al.* in [89] proposed another inversion algorithm which was also based on exponent decomposition. The main idea is as follows: Since

$$2^m - 2 = 2^{m-1} + 2^{m-1} - 2 = 2^{m-1} + 2^{m-2} .. + 2^{m-h} + 2^{m-h} - 2,$$

$$a^{-1} = a^{2^m-2} = a^{2^{m-1}}.a^{2^{m-2}}...a^{2^{m-h}+2^{m-h-2}}$$

Hence, the algorithm can use the inversion algorithm in [87] by replacing m by $(m - h)$. This method reduces the number of multiplications through performing an exhaustive search for an optimal value of h . The time complexity, however, is $O(\log_2(m))$.

4.2.3 Exponent Grouping Inversion Algorithms

Grouping exponent terms is another approach which attracted some researchers. However, the resulting speed is $O(m)$ which is worse than the $O(\log_2(m))$ of the Itoh and Tsujii inverter [87]. The inversion algorithms in this category are based on the idea of grouping exponent terms as follows:

Since $a^{-1} = a^{2^m-2} = (a^2).(a^4)...(a^{2^{m-1}})$, this allows grouping exponent terms in different ways. Fenn *et. al.* in [90] proposed an inversion algorithm which requires $\frac{m}{2}$ multiplications. The basic idea was by dividing the exponent terms into two equal groups. The first group contains $a^2, a^4, ..., a^k$, where $k = 2^{\frac{m-1}{2}}$. The other group contains the remaining terms till a^{2^m} . By multiplying the first term in each

group and repeated squaring by 2^{2^i} , the following formula can be applied to give the inverse within $\frac{m}{2}$ multiplications.

$$a^{-1} = \begin{cases} \prod_{i=1}^{\frac{m-1}{2}} [a \cdot a^{2^{\frac{m-1}{2}}}]^{2^i} & m \text{ odd;} \\ a^{2^{m-1}} \cdot \prod_{i=1}^{\frac{m-1}{2}} [a \cdot a^{2^{\frac{m-1}{2}}}]^{2^i} & m \text{ even.} \end{cases}$$

A further improvement, however, have been reported by Calvo and Torres [91]. The Calvo and Torres [91] inversion algorithm uses a fixed seed $a^2 \cdot a^4 = a^6$, which uses the same idea of grouping into two groups but in a different manner. This can be shown as:

$$a^{-1} = \begin{cases} \prod_{i=0}^{\frac{m-3}{2}} [a^6]^{2^{2^i}} & m \text{ odd;} \\ a^{2^{m-1}} \cdot \prod_{i=0}^{\frac{m-4}{2}} [a^6]^{2^{2^i}} & m \text{ even.} \end{cases}$$

This method was generalized by choosing m' and b such that $m = bm' + r$, where $r = [1, 2]$. This is basically the same way trying to group more terms in the seed at the beginning.

$$a^{-1} = \begin{cases} \prod_{i=0}^{b-1} [a^{2^{m'-1}-2}]^{2^{m'i}} & m = am' + 1 ; \\ a^{2^{m-1}} \cdot \prod_{i=0}^{b-1} [a^{2^{m'-1}-2}]^{2^{m'i}} & m = am' + 2. \end{cases}$$

This requires around $\frac{m}{m'}$ multiplications but still with the same $O(m)$ time complexity.

Yen proposed another approach for grouping terms which results in reduced number of clock cycles [92], . Yen realized that the fundamental idea behind Fenn's

et. al. inverter is to rearrange all terms into $\frac{m-1}{2}$ groups and to extract the common term $(a.a^{2^{\frac{m-1}{2}}})$. Accordingly, Yen redefined the common term to contain p terms and a^{-1} can be found as:

$$\prod_{i=1}^{k=\frac{m-1}{p}} (X)^{2^i}$$

where the common part X is precomputed as:

$$\prod_{j=0}^{p-1} a^{2^j \frac{m-1}{p}}$$

The best case requires $k + (p - 1)$ multiplications while the worst case requires $k + (p - 1) + (p - 2) = k + 2p - 3$ multiplications and the optimal selection of p for $m = pk + 1$ is \sqrt{m} . The time complexity, however, is still $O(m)$.

Table 4.3 summarizes the inversion algorithms and their time complexities. It is clear from Table 4.3 that exponent decomposition inversion algorithms provide the best time complexity among other inversion classes which require $O(m)$ multiplications.

4.3 Summary

In this chapter a brief survey of finite field arithmetic using normal basis over $GF(2^m)$ has been presented. Addition in normal basis requires simple XOR operation while squaring requires only a cyclic shift. This is the most attractive property of using normal basis as compared to using polynomial basis.

Table 4.3: The inverters and their time complexities.

Inverter	Class	Time Complexity (# of Multiplications)
Wang <i>et. al.</i> [78]	Standard	$(m - 2)$ multiplications
Itoh and Tsujii [87]	Exponent Decomposing	$\log_2(m - 1) + v(m - 1) - 1$
Feng [88]	Exponent Decomposing	$\lfloor \log_2(m) \rfloor + p$
Takagi <i>et. al.</i> [89]	Exponent Decomposing	$\log_2(m)$
Fenn <i>et. al.</i> [90]	Exponent Grouping	$\frac{m}{2}$
Calvo and Torres [91]	Exponent Grouping	$\frac{m}{2}$
Yen [92]	Exponent Grouping	$\frac{m-1}{p} + (p - 1)$

Normal basis multipliers are categorized in this dissertation into two main categories: (1) λ -matrix based multipliers, and (2) Conversion based multipliers. The conversion based multipliers are more efficient since they don't require storing the λ -matrix. The comparisons show that the Type II Sunar-Koc multiplier is the best multiplier with a hardware complexity of m^2 AND gates + $3/2 m(m - 1)$ XOR gates and a time complexity of $T_A + (1 + \lceil \log_2 m \rceil)T_X$. Accordingly, the Sunar-Koc multiplier has been selected for use in the cryptoprocessors proposed for implementations in this dissertation.

Inversion algorithms, on the other hand, are categorized into three main categories: (1) Standard, (2) Exponent Decomposing, and (3) Exponent Grouping. Exponent grouping inversion algorithms have better performance compared to the standard with time complexity $O(m)$. The exponent decomposing inversion algorithm of Itoh and Tsujii, however, is found to be the best inverter requiring only $\log_2(m - 1)$ multiplications. Accordingly, the Itoh-Tsujii inverter has been chosen

for the implementations presented in this dissertation.

Chapter 5

Power Analysis Attacks and $GF(2^m)$ FPGA Implementations

In this chapter we survey work on power analysis attacks. This chapter also surveys existing elliptic curve cryptosystem implementations on FPGA over $GF(2^m)$ using normal basis.

5.1 Power Analysis Attacks

Power analysis attacks are usually divided into two types. The first type, Simple Power Analysis (SPA), is based on a single observation of power consumption, while the second type, Differential Power Analysis (DPA) combines SPA attack with an error-correcting technique using statistical analysis [11]. More importantly, classical

DPA attacks have been extensively researched for each cryptosystem and new types of DPA are continuously being developed. Many of the existing countermeasures are vulnerable to the more recent attacks which include Refined Power Analysis (RPA) [94], Zero Power Analysis (ZPA) [95], Doubling Attack [96] and Address-Bit Differential Power Analysis (ADPA) [102]. In the next subsections, these attacks are described in more detail.

5.1.1 Simple Power Analysis

A SPA attack consists of observing the power consumption during a single execution of a cryptographic algorithm. The power consumption analysis may also enable one to distinguish between point addition and point doubling in the double-and-add algorithm.

Coron [97] showed that for Algorithm 3.1 to be SPA resistant, the instructions performed during a cryptographic algorithm should not depend on the data being processed, e.g. there should not be any branch instructions conditioned by the data. This could be done by performing the addition and doubling each time and then at the end of the loop decide whether to accept the result or to eliminate the addition part according to k_i value (see Algorithm 5.1). However, even though this scheme is resistant to a SPA attack, it remains vulnerable to a DPA attack.

Algorithm 5.1 Double-and-add-always Scalar Multiplication Algorithm.

1. input P, k
 2. $Q[0] \leftarrow P$
 3. for i from $m - 2$ to 0 do
 - 3.1. $Q[0] \leftarrow 2Q[0]$
 - 3.2. $Q[1] \leftarrow Q[0] + P$
 - 3.3. $Q[0] \leftarrow Q[k_i]$
 4. output $Q[0]$
-

5.1.2 Differential Power Analysis

A DPA attack is based on the same basic concept as a SPA attack, but uses error correction techniques and statistical analysis to extract very small differences in the power consumption signals. To be resistant to a DPA attack, some system parameters or computation procedures must be randomized. Coron suggested three countermeasures to protect against DPA:

1. Randomization of the private exponent: Let $\#E$ be the number of points of the curve. The computation of $Q = kP$ is done as follows:
 - Select a random m -bit number d .
 - Compute $k' = k + d \#E$.
 - Compute the point $Q = k'P$. We have $Q = kP$ since $\#EP = \mathcal{O}$.
2. Blinding Point P : The point P to be multiplied is “blinded” by adding a secret random point R for which we have know $S = kR$. Scalar multiplication is done by computing the point $k(R + P)$ and subtracting $S = kR$ to get $Q = kP$.

3. Randomized Projective Coordinates: The projective coordinates of a point are not unique because:

$$(X, Y, Z) = (\lambda X, \lambda Y, \lambda Z) \quad (5.1)$$

for every $\lambda \neq 0$ in the finite field. The third countermeasure randomizes the projective coordinate representation of a point $P = (X, Y, Z)$. Before each new execution of the scalar multiplication algorithm for computing $Q = kP$, the projective coordinates of P are randomized with a random value λ . The randomization can also occur after each point addition and doubling.

An enhanced version of Coron's 3rd countermeasure has been proposed by Joye and Tymen [98]. It uses an isomorphism of an elliptic curve, thereby transposing the computation into another curve through a random morphism. The elliptic point $P = (X, Y, Z)$ and parameters (a, b) of the defined curve equation can be randomized like $(\lambda^2 X, \lambda^3 Y, Z)$ and $(\lambda^4 a, \lambda^6 b)$. However, all of the above countermeasures add computational overhead and are still vulnerable to differential power attacks as described below.

Doubling Attack

The doubling attack obtains the secret scalar using binary elliptic scalar multiplication [96]. It only works for the most-to-least version of the double-and-add algorithm. The main idea of this attack is based on the fact that, even if an adver-

sary cannot see whether the computation being done is doubling or addition, he can still detect when the same operation is done twice. More precisely, if $2A$ and $2B$ are computed in any operations, the attacker is not able to guess the value of A or B but he can check if $A = B$ or $A \neq B$. This assumption is reasonable since this kind of computation usually takes many clock cycles and depends greatly on the value of the operands. If the noise is negligible, a simple comparison of the two power traces during the doubling will be efficient to detect this equality.

Two of Coron's three proposed countermeasures against DPA attacks fail to protect against a doubling attack: randomizing the private scalar (exponent) and blinding the point. However, his third countermeasure, the randomized projective coordinate does protect against a doubling attack as does a randomized exponentiation algorithm such as the Ha and Moon algorithm [99] which maps a given scalar to one of various representations. Since the positions of the zeros in the Ha and Moon algorithm vary in each representation, the doubling attack cannot detect the positions of the zeros for the doubling operation.

Basically, to protect against a doubling attack, the random blinding point R should be randomly updated. A regularly updated method shouldn't be chosen. A method similar to Coron's 3rd countermeasure or a random field isomorphism should be used.

RPA and ZPA Attacks

Goubin proposed a new power analysis in 2003, namely the refined power analysis (RPA), which works even if one of the three countermeasures with a SPA countermeasure is applied [94]. The RPA attack assumes that the attacker can input adaptively chosen messages or elliptic curve points to the victim exponentiation algorithm. Smart [100] analyzed the RPA attack in detail and discounted its effectiveness in a large number of order. For the remaining cases Smart proposed a defense against the RPA attack based on isogenies of small degree [100]. However, the RPA attack is still a threat to most elliptic curve cryptosystems.

The zero-value point attack is an extension of the RPA attack [95]. In a RPA attack, the attacker uses a special point which has a zero-value coordinate. In a ZPA attack, on the other hand, an attacker utilizes an auxiliary register which might take a zero-value in the definition field. As a result, Coron's 3rd or random field isomorphism countermeasures do not protect against ZPA attacks.

To protect against RPA and ZPA attacks, the base point P or the secret scalar k should be randomized. For example, Coron's first two countermeasures protect against these attacks. Mamiya *et. al.* [101] recently proposed a countermeasure, called BRIP, which uses a random initial point R . The proposed countermeasure computes $kP + R$ and then subtracts R to get kP . Thus, no special point or zero-value register will appear during all operations and hence it is resistant against both

RPA and ZPA attacks.

Address-Bit Differential Power Analysis Attack

In 1999, Messerges *et. al.* proposed a new attack against secret key cryptosystems, the address-bit DPA (ADPA), which analyzes a correlation between the secret information and addresses of registers [102]. Itoh *et. al.* in 2002 extended this attack to Elliptic Curve based Cryptosystems [103]. Basically, ADPA Attack is based on the correlation between bit values of the scalar and the location (address) of the variables used in a scalar multiplication algorithm. The countermeasures used to protect against simple power analysis and differential power analysis that are based on randomization of the base point or the projective coordinate do not provide countermeasure against address-bit analysis attacks. Therefore, these countermeasures do not remove the correlation between the bit values of a scalar and the location (address) of the variables used in a scalar multiplication algorithm.

A hardware-based DPA countermeasure proposed by May *et. al.* [104] is based on Randomized Register Renaming (RRR). RRR is supposed to be implemented on a processor called NDISC, which can execute instructions in parallel. In other words, it requires a special hardware to work. Itoh *et. al.* gave several countermeasures against the ADPA attack in [103]. But those countermeasures double the computing time.

In 2003, Itoh *et. al.* proposed a countermeasure [105], called randomized address-

ing method (RA), which is similar to RRR but does not require special hardware. In RA, the addresses of registers are randomized by a random number for each scalar exponentiation. Thus, all addresses of registers are randomized and hence the side channel information are also randomized.

5.2 $GF(2^m)$ FPGA Implementations

Reconfigurable FPGAs were used in [21,22,25–36,38–40,42,44–47]. These implementations can be classified according to their arithmetic basis representations into two classes: Optimal Normal Basis (ONB) based implementations [22, 25, 27, 30, 36, 45, 47] and Polynomial Basis (PB) based implementations [21,26,28,29,31–35,38–40,42–44]. We are interested in optimal normal basis implementations only which are described briefly below.

The first implementation of $GF(2^m)$ elliptic curves on FPGAs using optimal normal basis was reported by Gao *et. al.* [22] in 1999 (Figure 5.1). Two FIFOs (Figure 5.1) are used to serve as input/output buffers and the dual-port register file is used to save input parameters and intermediate data. Gao *et. al.* [22] implemented the same multiplier that Gao has developed earlier [80], while the implemented inverter was that of Itoh and Tsujii [87]. The implementation was carried out on a Xilinx XC4044XL FPGA with $m = 53$ bits.

Leung *et. al.* [25] reported an elliptic curve cryptosystem implementation on

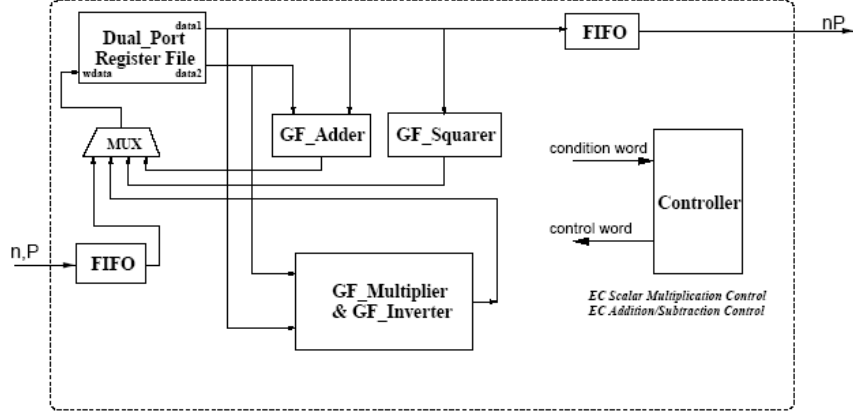


Figure 5.1: Gao *et. al.* ECC coprocessor [22].

a Xilinx FPGA XCV300 device with $m = 113, 155, 281$ bits. Point operations were implemented as sequences of micro-coded field operations. This allows many algorithmic optimization without changing the hardware. The entire design was described by a module generator, which is a program written in Perl that takes the key size m as an input parameter and produces a VHDL code of the elliptic curve cryptoprocessor as an output. Thus, an arbitrary size of the cryptosystem can be generated.

Leong *et. al.* [30] have reported another cryptoprocessor with a parallel version of the field multiplier presented in [25] using a Xilinx Virtex XCV1000 FPGA device. The architecture of the cryptoprocessor used in [25] and [30] is shown in Figure 5.2.

Ernst *et. al.* presented a generator-based design and validation methodology for rapid prototyping of an elliptic curve cryptosystem hardware (Figure 5.3) [27].

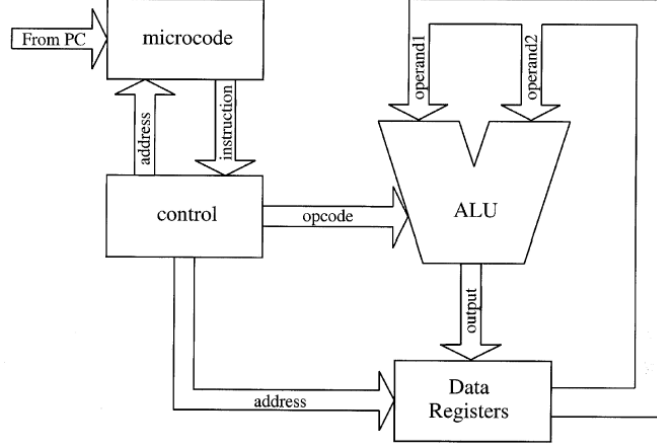


Figure 5.2: ECC processor architecture used by Leung *et. al.* [25] and Leong *et. al.* [30]

A generator program accepts the two main parameters, key size and multiplier radix, and generates a highly efficient RTL description, which can be synthesized onto an FPGA. This approach allows the design to effortlessly exploit the available resources on the FPGA for variable security and performance requirements. The proposed generator approach on top of a VHDL based design flow has lead to a 270-bit Cryptoprocessor design including three Massey and Omura serial multipliers. The implementation used Xilinx FPGA XC4085XLA with $m = 151, 191$ and 270 bits.

The use of hybrid projective coordinates and hybrid basis representations over $GF(2^m)$ was proposed by Bednara *et. al.* [36]. The main idea is to utilize attractive features such as mixed coordinates as well as either of the polynomial or the normal basis. The structure of the datapath architecture used is shown in Figure 5.4.

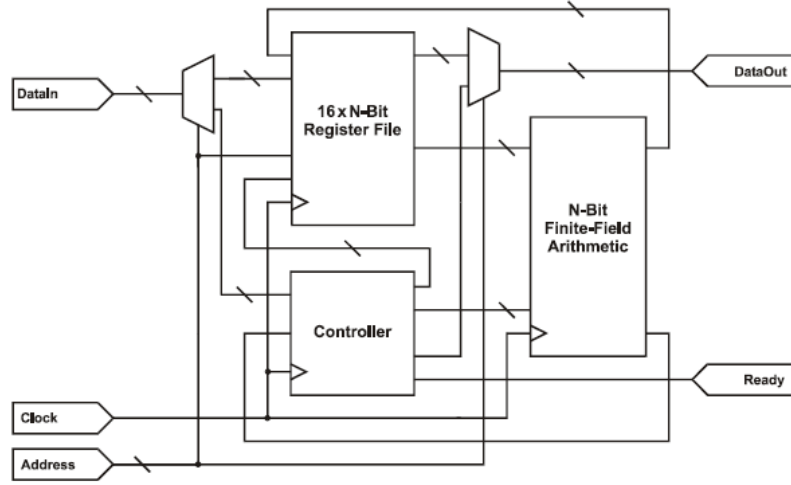


Figure 5.3: Ernst *et. al.* ECC architecture [27].

Bednara *et. al.* used two squarers, two adders and four sequential multipliers. Each of these arithmetic units (AU) can get operands from a dual-port operand memory, a register, or directly from the output of another arithmetic unit. The arithmetic control unit (ACU) generates control signals for all AUs, the operand memory and the register. The second port of the operand memory is used by a host interface, thus allowing for host data transfer while a point multiplication is being performed. The AUs consist of one or two operand registers, one output register and the core functional unit. The prototype implementation used a Xilinx FPGA XCV1000 with $m = 191$ bits, Massy-Omura multiplier for normal basis and LFSR multiplier for polynomial basis. The question arises here as to how efficient is such hybrid system since we need conversion modules between polynomial and normal basis and vice versa.

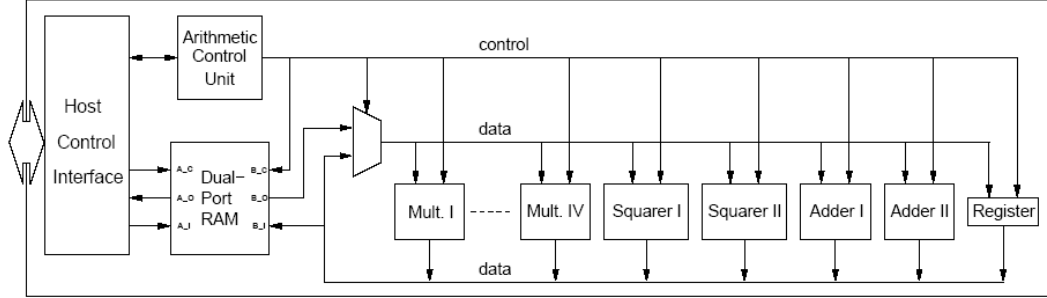


Figure 5.4: Bednara *et. al.* ECC architecture [36].

Cheung *et. al.*, presented a design generator for producing hardware designs for elliptic curve cryptosystems over the finite field $GF(2^m)$ [45]. The design generator can automatically produce implementations with different speed, size and level of security. The major customizable elements of a cryptosystem are: the key size, the degree of parallelism, and the protocols of the system. Parallel field multipliers were used to exploit the inherent parallelism within point operations in projective coordinate [16] to speedup scalar multiplication. The architecture of the proposed cryptoprocessor is shown in Figure 5.5. The cryptoprocessor was implemented on Xilinx FPGA XC2V6000 with $m = 113$ and 270 bits.

Recently, Al-Somani and Ibrahim in [47], presented an elliptic curve cryptoprocessor with resistance against timing attacks [93]. The proposed cryptoprocessor is based on optimal normal basis representation and uses three multipliers to perform parallel field multiplications as shown in Figure 5.6. Point operations are performed using Mixed coordinate system to increase the performance and the im-

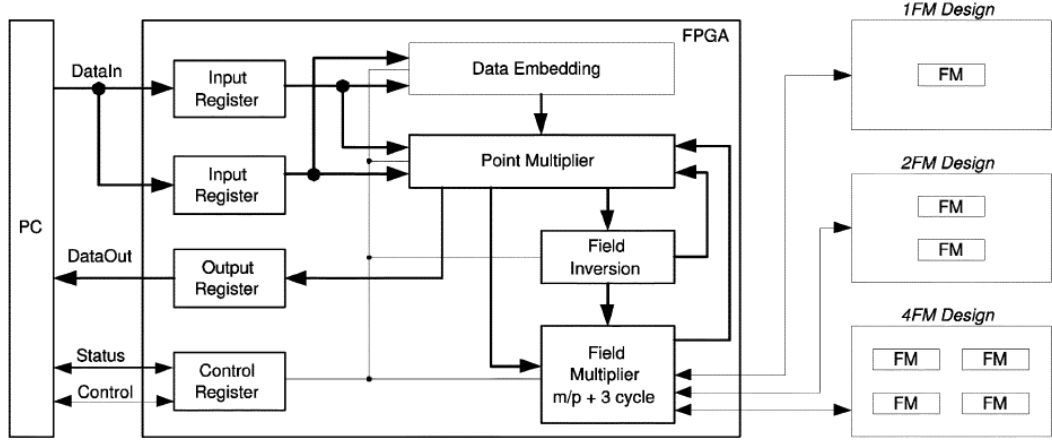


Figure 5.5: Cheung *et. al.* ECC architecture [45].

munity against timing attacks. The basic idea was to select a combination of point addition and point doubling from Mixed coordinate system such that both point operations take the same number of multiplication cycles. Thus, an attacker cannot distinguish between point doubling and point addition and therefore it is not possible to extract the key pattern using a timing attack. The implementation used Xilinx XC2V8000 FPGA with $m = 173$ bits.

5.3 Remarks on the reviewed implementations

Reviewing these FPGA based implementations, several remarks need to be pointed out:

- Most of the existing implementations were for prototyping purpose and FPGA was only used as a vehicle.

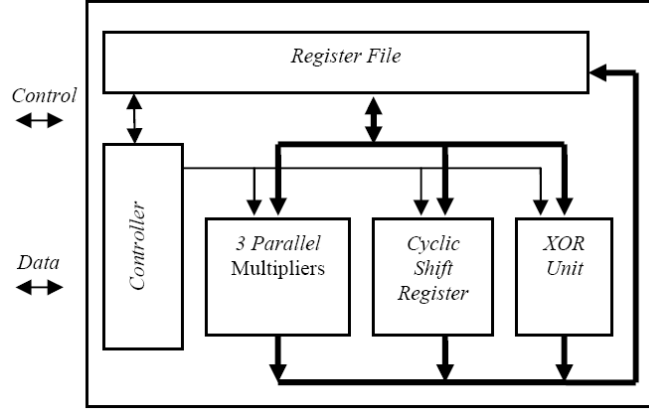


Figure 5.6: Al-Somani and Ibrahim ECC architecture [47].

- All of the existing implementations used existing FPGAs and none of them proposed new FPGA architecture that targets such applications for higher performance.
- Most of the implementations use polynomial basis (PB) representation, while it is well known that optimal normal basis (ONB) is more suitable for hardware implementations than polynomial basis.
- The comparisons between these implementations is somewhat difficult because of their different objectives, constraints and FPGA implementation technology.

Table 5.1 summarizes reported ECC implementations on FPGAs over $GF(2^m)$. It is clear from Table 5.1 that only one polynomial and another optimal normal basis implementations among those reported have provided resistance to some power analysis attack.

Table 5.1: $GF(2^m)$ Implementations on FPGAs.

Ref. No.	Year	Platform	Rep.	Power Analysis Attacks Resistance
Rosner [21]	1998	Xilinx XC4062	PB	–
Gao [22]	1999	Xilinx XC4044XL	ONB	–
Leung [25]	2000	Xilinx XCV300	ONB	–
Orlando [26]	2000	Xilinx XCV400E	PB	–
Ernst [27]	2001	Xilinx XC4085XLA	ONB	–
Smart [28]	2001	Xilinx XC4000XL	PB	–
Orlando [29]	2002	Xilinx XCV1000E	PB	–
Leong [30]	2002	Xilinx XCV1000	ONB	–
Ernst [31]	2002	Atmel AT94K40	PB	–
Gura [32, 33]	2002	Xilinx XCV2000E	PB	–
Jung [34]	2002	Atmel AT94K40	PB	–
Kerins [35]	2002	Xilinx XCV2000	PB	–
Bednara [36]	2002	Xilinx XCV1000	PB	–
Lutz [38]	2004	Xilinx XCV2000E	PB	–
Mentens [39]	2004	Xilinx XCV800	PB	–
Jarvinen [40]	2004	Xilinx XC2V8000	PB	–
Saqib [42]	2004	Xilinx XCV3200	PB	–
Batina [44]	2005	Xilinx XCV800	PB	Simple Power Attacks
Cheung [45]	2005	Xilinx XC2V6000	ONB	–
Al-Somani [47]	2006	Xilinx XC2V8000	ONB	Timing Attacks

5.4 Summary

In this chapter, power analysis attacks and their existing countermeasures have been described. Power analysis attacks are usually divided into two types: (1) simple power analysis attacks, and (2) differential power analysis attacks. Simple power analysis attacks are based on a single observation of power consumption.

Differential power analysis attacks, on the other hand, combines simple power analysis attacks with error-correcting techniques using statistical analysis tools. Many of the existing countermeasures are vulnerable to the more recent differential power analysis attacks which include (1) refined power analysis attack, (2) zero power analysis attack, (3) doubling attack, and (4) address-bit differential power analysis attack.

Elliptic curve cryptosystem implementations on FPGA over $GF(2^m)$ using normal basis have been also surveyed in this chapter. Only one polynomial and another optimal normal basis implementations among those reported have provided resistance to some power analysis attack.

Chapter 6

Secure ECC Cryptoprocessor

Architectures

Several elliptic curve cryptoprocessors have been proposed and implemented on FPGAs using optimal normal basis over $GF(2^m)$. None of the reported implementations provides security against all known power analysis attacks. In this dissertation we propose two $GF(2^m)$ ECC cryptoprocessors that are secure against all known DPA attacks. One cryptoprocessor sequentially processes randomized key partitions in random order, while the other is a parallel cryptoprocessor with each key partition processed by an independent scalar multiplier.

The merits of these two cryptoprocessors are compared to a regular sequential ECC single processor which is used as a reference for such comparison. The following sections provide details of these three elliptic curve architectures; namely: (1)

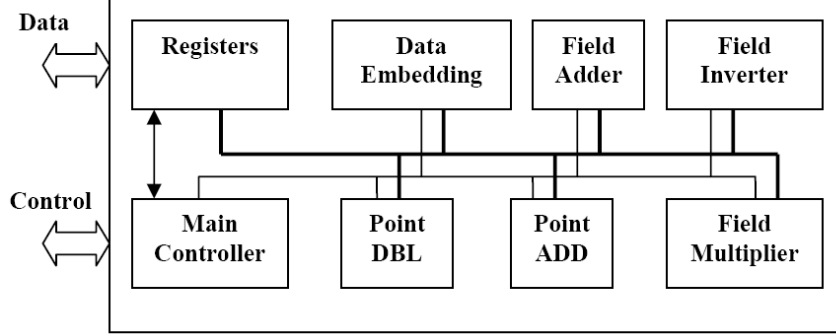


Figure 6.1: The proposed architecture

ECC_{NS} : a non-secure elliptic curve cryptoprocessor architecture used as a reference design for comparison, (2) ECC_{SS} : the sequential cryptoprocessor architecture with resistance against power analysis attacks and (3) ECC_{PS} : the parallel cryptoprocessor architecture with resistance against power analysis attacks.

6.1 The ECC_{NS} Cryptoprocessor

This section presents the architecture of a regular $GF(2^m)$ elliptic curve cryptoprocessor which does not provide security against power analysis attacks. The proposed cryptoprocessor architecture is modeled using VHDL and is fully parameterized. The basic units of this architecture are: (1) the main controller, (2) the data embedding unit, (3) the point addition and doubling units and (4) the field arithmetic units (adder, multiplier and inverter). In the following subsections, these units are described in detail (Figure 6.1).

6.1.1 Main Controller

The double-and-add algorithm has been selected for scalar multiplication (Algorithm 3.1). For the encryption/decryption process, the selected encryption protocol is the elliptic curve Diffie-Hellman protocol. The pseudocode of the ECC_{NS} cryptoprocessor is given in Algorithm 6.1.

The inputs of Algorithm 6.1 are: (1) the base point P , (2) the elliptic curve parameters a , b , (3) the secret key k , (4) the encryption/decryption mode and (5) the plaintext/ciphertext. The output is either the ciphertext or the plaintext depending on the encryption/decryption mode.

Referring to the cryptoprocessor pseudocode (Algorithm 6.1, scalar multiplication starts at Step 1 by executing the double and add algorithm. The encryption process starts at Step 2 by embedding the plaintext into a random point on the elliptic curve. The scalar multiplication result (kP) is added to this point to produce a ciphered point. The decryption process (Step 3), however, subtracts (kP) from the ciphered point.

6.1.2 Data Embedding

Data embedding is performed within the x -coordinate of a point on the elliptic curve. A random number is picked to fill the 5 most significant bits and the remaining bits will contain the data to be encrypted. If this x -coordinate is not a valid point on

Algorithm 6.1 Pseudocode of the ECC_{NS} cryptoprocessor.

Inputs: P : Base Point, k : Secret key, a , b : Elliptic curve parameters.

Plaintext/Ciphertext, Encryption/Decryption.

Outputs: Ciphertext/Plaintext.

Scalar Multiplication (kP):

1. Algorithm 3.1(P , k).

Encryption/Decryption Process:

2. if (Encrypt) then
 - 2.1. Embed the plaintext in random points on the elliptic curve.
 - 2.2. ADD (kP) to data points.
 - 2.3. Output (ciphertext).
 3. else
 - 3.1. ADD ($-kP$) to ciphered points.
 - 3.2. Extract the plaintext from the data points.
 - 3.3 Output (plaintext).
-

the elliptic curve, another random number is picked until a valid elliptic curve point is obtained. The checking procedure is as follows [106]:

- Recall the elliptic curve equation defined over $GF(2^m)$:

$$y^2 + xy = x^3 + ax^2 + b \quad (6.1)$$

where $a, b \in GF(2^m)$ and $b \neq 0$.

- Rewrite Equation 6.1 as

$$y^2 + xy + f(x) = 0 \quad (6.2)$$

where $f(x) = x^3 + ax^2 + b$.

- Let $y = zx$, Equation 6.2 becomes:

$$z^2 + z + c = 0 \tag{6.3}$$

where

$$c = f(x) \cdot x^{-2} \tag{6.4}$$

- Find the trace of c , the trace function is simply the parity function which can be easily implemented by computing the XOR of all the bits.
- If the trace is 1, try another random number and repeat the check again. If the trace is 0, this is a valid x -coordinate and proceed to recover the y -coordinate.
- By taking the square root of Equation 6.3, it can be rewritten as:

$$z^{1/2} = z + c^{1/2} \tag{6.5}$$

which can be also rewritten as:

$$z_i = z_{i-1} + c_i \tag{6.6}$$

- Since $z + 1$ is actually the complement of z in a normal basis, in one of the two solutions the least significant bit will be 0 and the other one will be 1. We then further compute all the other bits one by one.

- To compute the y value, simply multiply z by x .

6.1.3 Point Addition and Doubling

Point addition and doubling are performed using Lopez-Dahab projective coordinate system which takes the form $(x, y) = (X/Z, Y/Z^2)$ [64]. Point addition and point doubling require only 14 and 5 field multiplications respectively (Table 2.3). The projective elliptic curve equation of the affine Equation (6.1) is given by

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4 \quad (6.7)$$

If $Z = 0$ in Equation 6.7, then $Y^2 = 0$, i.e., $Y = 0$. Therefore, $(1, 0, 0)$ is the only projective point that satisfies the equation for $Z = 0$. This is the point at infinity \mathcal{O} [64]. To convert an affine point (x, y) into Lopez-Dahab projective coordinate, set $X = x$, $Y = y$, $Z = 1$. Similarly, to convert a projective point back to affine coordinate, we compute $x = X/Z$, $y = Y/Z^2$. The additive inverse of a point $P = (X, Y, Z)$ is the point $(X, XZ + Y, Z)$ which is used at the end of the decryption process [59].

The projective point operations formulas of the Lopez-Dahab coordinate system [64] has been reported only for the most-to-least version of the scalar multiplication algorithm. Alternatively, point doubling and point addition formulas that are suitable for both versions of the scalar multiplication algorithm are proposed here

(Table 6.1). Clearly, the doubling formula requires only 5 field multiplications, 5 field squarings and 5 storage registers. The point addition formula requires 14 field multiplications, 6 field squarings and 8 storage registers.

6.1.4 Field Operations

One key advantage of optimal normal basis representation is the simplicity of the squaring operation. Field squaring is simply a cyclic shift operation. Field addition is a Boolean XOR operation and is implemented using an m -bit XOR unit. Thus, only one clock cycle is required to perform either of the two operations, i.e., field squaring or field addition.

Multiplication is more complicated than addition and squaring. An efficient multiplier is highly needed and is the key for efficient finite field computations. The Sunar–Koc multiplier has been selected since it provides the best space and time complexities reported thus far [84].

The main idea of Sunar–Koc multiplier is based on converting the two operands to equivalent representations in another shifted basis, performing the multiplication in that basis and converting the product back to the normal basis. The conversion step requires only a single clock cycle since it is nothing but a permutation of the normal basis. As explained in Chapter 4, two elements \hat{A} and $\hat{B} \in GF(2^m)$ are represented in the shifted basis as:

$$\hat{A} = \sum_{i=1}^m \hat{a}_i(\gamma^i + \gamma^{-i}) = \sum_{i=1}^m \hat{a}_i\beta_i, \text{ and } \hat{B} = \sum_{i=1}^m \hat{b}_i(\gamma^i + \gamma^{-i}) = \sum_{i=1}^m \hat{b}_i\beta_i$$

The product $\hat{C} = \hat{A} \cdot \hat{B}$ is written as:

$$\begin{aligned} \hat{C} &= \left(\sum_{i=1}^m \hat{a}_i(\gamma^i + \gamma^{-i}) \right) \left(\sum_{j=1}^m \hat{b}_j(\gamma^j + \gamma^{-j}) \right) \\ &= \sum_{i=1}^m \sum_{j=1}^m \hat{a}_i \hat{b}_j (\gamma^{i-j} + \gamma^{-(i-j)}) + \sum_{i=1}^m \sum_{j=1}^{m-i} \hat{a}_i \hat{b}_j (\gamma^{i+j} + \gamma^{-(i+j)}) \\ &\quad + \sum_{i=1}^m \sum_{j=m-i+1}^{m-i} \hat{a}_i \hat{b}_j (\gamma^{2m+1-(i+j)} + \gamma^{-(2m+1-(i+j))}) \\ &= \hat{C}_1 + \hat{D}_1 + \hat{D}_2 \end{aligned}$$

Sunar and Koc [84] precompute and store the results of $\hat{a}_i \hat{b}_j \forall i, j \in 1, 2, \dots, m$ which requires m^2 storage bits. Combination of these bits are then added modulo 2 using XOR gates to generate the m product bits. Assuming that the operands and the resulting product are registered, $3m$ storage bits will be further required. Thus, the space complexity of Sunar–Koc parallel multiplier is m^2 AND gates + $\frac{3}{2}(m^2 - m)$ XOR gates + $(m^2 + 3m)$ storage bits.

Since we are using FPGA as implementation technology to evaluate our proposed architectures, we have opted for implementing a sequential version of the Sunar–Koc multiplier to save on available FPGA resources. The proposed sequential multiplier requires only three barrel shifters and three other registers alleviating the need for precomputing and storing $\hat{a}_i \hat{b}_j \forall i, j \in 1, 2, \dots, m$. The dataflow of the sequential

multiplier is shown in Figure 6.2.

In Figure 6.2, the two operands A and B are passed to the conversion box to convert A and B from normal basis to the shifted basis. The conversion box is used for (1) converting the two operands A and B to the shifted basis and (2) converting the product \hat{C} , which is represented in the shifted basis, back to normal basis.

The pseudocode of the conversion box is given in Algorithm 6.2. To convert from normal basis to the shifted basis and from the shifted basis to normal basis, the β conversion vector is generated to keep the permutation order (Steps 1-2). The β vector depends only on m and is computed only once at system startup. Steps 3-4 show the conversion process of the two operands A and B from normal basis to the shifted basis. The conversion process of the product \hat{C} from the shifted basis back to normal basis is performed at Steps (5-6).

The circuits of the \hat{C}_1 , \hat{D}_1 and \hat{D}_2 units are shown in Figures 6.3, 6.4 and 6.5 respectively. Figure 6.3 shows that we only need 2 AND gates and 1 XOR gate to produce a bit to the barrel shifter. Accordingly, $(m - 1)$ copies of these gates are required to produce the required bits to the barrel shifter in each iteration. The barrel shifter is controlled by the index i which is incremented by one each clock cycle. In each clock cycle, the barrel shifter content is accumulated into the \hat{C}_1 register. A total of $(m - 1)$ clock cycles is required for \hat{C}_1 to be computed. The pseudocode of \hat{C}_1 is given in Algorithm 6.3.

The circuit of \hat{D}_1 (Figure 6.4) is much simpler than \hat{C}_1 since it requires only

Algorithm 6.2 Pseudocode of the conversion box unit.

Inputs: $A, B/\widehat{C}$, to-shifted-basis/to-normal-basis.

Outputs: $\widehat{A}, \widehat{B}/C$.

β Conversion vector:

1. for i in 0 to $m - 1$ do
 - 1.1. if $(2^i \leq m)$ then
 - 1.1.1. $\beta_i = 2^i$
 - 1.2. else if $(2^i \bmod 2m + 1) \leq m$ then
 - 1.2.1. $\beta_i = 2^i \bmod 2m + 1$
 - 1.3. else
 - 1.3.1. $\beta_i = 2m + 1 - 2^i \bmod 2m + 1$
2. end for

Conversion to Shifted Basis:

3. if (to-shifted-basis) then
 - 3.1. for i in 0 to $m - 1$ do
 - 3.1.1. $\widehat{A}_{\beta_i} = A_i$
 - 3.1.2. $\widehat{B}_{\beta_i} = B_i$
 - 3.2. end for
4. Output $(\widehat{A}, \widehat{B})$.

Conversion Back to Normal Basis:

5. if (to-normal-basis) then
 - 5.1. for i in 0 to $m - 1$ do
 - 5.1.1. $C_i = \widehat{C}_{\beta_i}$
 - 5.2. end for
 6. Output (C) .
-

1 AND to produce 1 bit for the barrel shifter which is controlled by the index i . Similarly, $(m - 1)$ copies of these gates are needed to produce the required bits to the barrel shifter in each iteration. It takes $(m - 1)$ clock cycles to compute \widehat{D}_1 . The circuit of \widehat{D}_2 (Figure 6.5) has the same circuitry as \widehat{D}_1 , but the difference is that it requires m copies of the gates and accordingly requires m clock cycles to complete. Since both \widehat{C}_1 and \widehat{D}_1 require $(m - 1)$ clock cycles, the content of \widehat{C}_1 and \widehat{D}_1 are added together and the result is added to the content of \widehat{D}_2 to produce \widehat{C} . The

pseudocode of \hat{D}_1 and \hat{D}_2 are given in Algorithms 6.4 and 6.5 respectively.

Finally, \hat{C} is passed to the conversion box to convert it from the shifted basis back to normal basis. Thus, the multiplication process requires two extra clock cycles for conversion in each multiplication; one clock cycle for converting A and B to the shifted basis and another clock cycle for converting \hat{C} back to normal basis.

The time complexity of the proposed sequential multiplier in the shifted basis is $(m-1)(T_A + 2T_X) + 2T_X$, while the space complexity is:

- m storage bits for the β conversion vector.
- $3m$ storage bits for the converted operands and result.
- $(2m-2)$ AND gates + $(2m-1)$ XOR gates + $2m$ storage bits for $C1$.
- $(m-1)$ AND gates + m XOR gates + $2m$ storage bits for $D1$.
- m AND gates + m XOR gates + $2m$ storage bits for $D2$.
- m XOR gates used between $C1$ and $D1$.
- m XOR gates used between $D2$ and shifted C register.

Thus, the space complexity of the proposed serial multiplier is $(4m-3)$ AND gates + $(6m-1)$ XOR gates + $10m$ storage bits.

The incryption/decryption process requires only one inversion since we are using projective coordinate, while an inversion per trial is required for data embedding

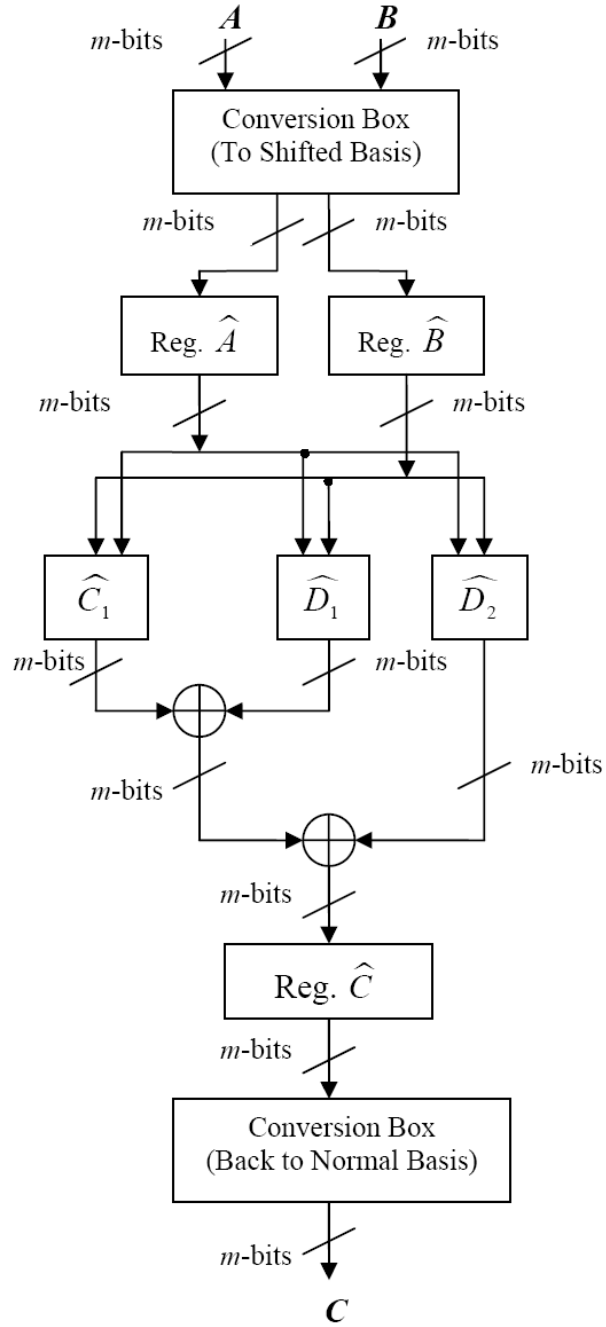


Figure 6.2: Dataflow of the proposed sequential multiplier.

in a valid x -coordinate (Equation 6.4). Thus, an efficient inverter is required. The selected inverter is the Itoh and Tsujii inverter [87]. The dataflow of the Itoh–Tsujii inverter is shown in Figure 6.6. Figure 6.6 shows that Itoh–Tsujii inverter requires three cyclic shift registers, one barrel shifter, one down counter and one multiplier (note that only one multiplier is used while two are drawn in the dataflow diagram for the purpose of clarity).

In Figure 6.6, the down counter s controls the barrel shifter r in each iteration. The barrel shifter r , accordingly, controls the required number of squarings by the cyclic shift register q . The least bit of the barrel shifter r_0 , on the other hand, decides if the multiplication of the content of the cyclic shift register t by a is required or not. The Itoh–Tsujii inversion algorithm is given in Algorithm 6.6. Clearly, the inverter depends a lot on the field multiplier. The Itoh–Tsujii inversion algorithm requires only $O(\log_2(m))$ multiplications, which is the best among other inversion algorithms reported thus far [76].

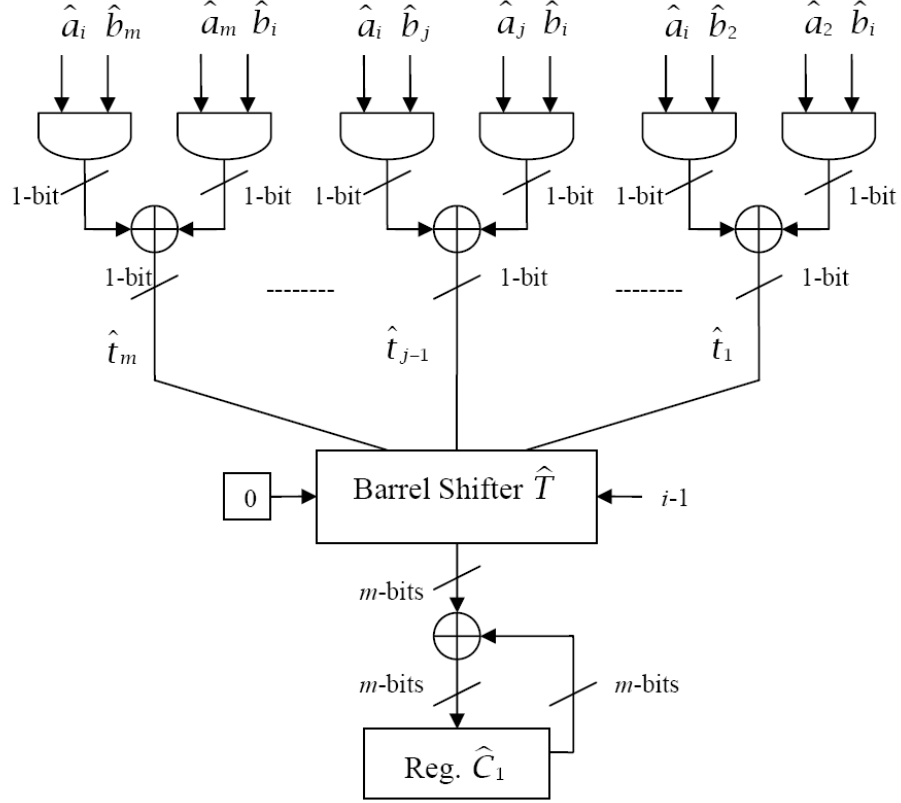


Figure 6.3: Circuit of $C1$, $i = [1, m - 1]$, $j = [2, m]$.

Algorithm 6.3 Pseudocode of \hat{C}_1 .

Inputs: \hat{A}, \hat{B} .

Outputs: \hat{C}_1 .

1. for i in 1 to $m - 1$ do
 - 1.1. for j in 1 to $m - 1$ do in Parallel
 - 1.1.1. $\hat{t}_{j-1} = (\hat{a}_i \text{ AND } \hat{b}_j) \text{ XOR } (\hat{a}_j \text{ AND } \hat{b}_i)$
 - 1.2. end for
 - 1.3. Shift Right \hat{T} by $i - 1$
 - 1.4. $\hat{C}_1 = \hat{C}_1 \text{ XOR } \hat{T}$
 2. end for
 3. Output (\hat{C}_1) .
-

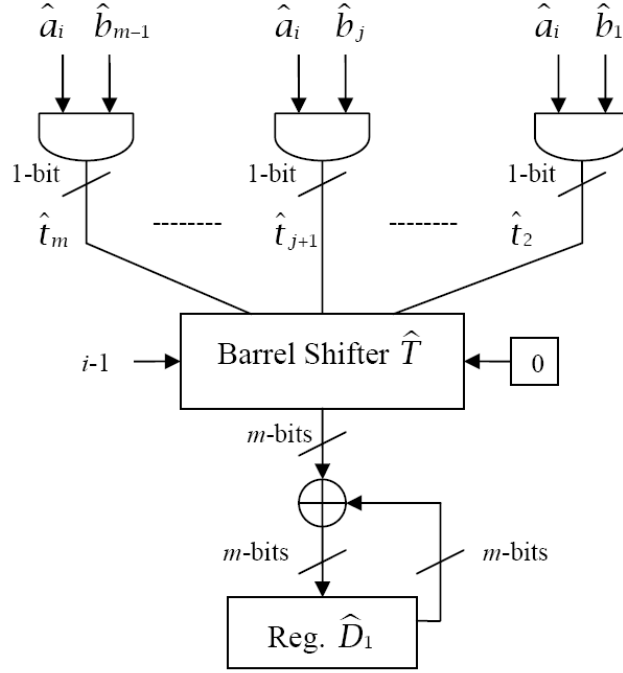


Figure 6.4: Circuit of $D1$, $i = [1, m - 1]$, $j = [1, m - 1]$.

Algorithm 6.4 Pseudocode of \hat{D}_1 .

Inputs: \hat{A}, \hat{B} .

Outputs: \hat{D}_1 .

1. for i in 1 to $m-1$ do
 - 1.1. for j in 1 to $m - 1$ do in Parallel
 - 1.1.1. $\hat{t}_{j+1} = \hat{a}_i \text{ AND } \hat{b}_j$
 - 1.2. end for
 - 1.3. Shift Left \hat{T} by $i - 1$
 - 1.4. $\hat{D}_1 = \hat{D}_1 \text{ XOR } \hat{T}$
 2. end for
 3. Output (\hat{D}_1) .
-

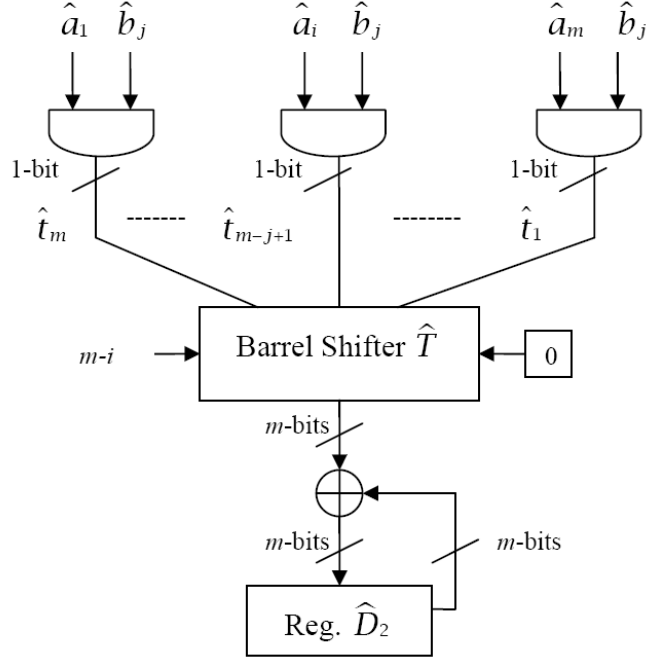


Figure 6.5: Circuit of D_2 , $i = [1, m]$, $j = [1, m]$.

Algorithm 6.5 Pseudocode of \hat{D}_2 .

Inputs: \hat{A} , \hat{B} .

Outputs: \hat{D}_2 .

1. for i in m to 1 do
 - 1.1. for j in m to 1 do in Parallel
 - 1.1.1. $\hat{t}_{m-j+1} = \hat{a}_j \text{ AND } \hat{b}_i$
 - 1.2. end for
 - 1.3. Shift Left \hat{T} by $m - i$
 - 1.3. $\hat{D}_2 = \hat{D}_2 \text{ XOR } \hat{T}$
 2. end for
 3. Output (\hat{D}_2).
-

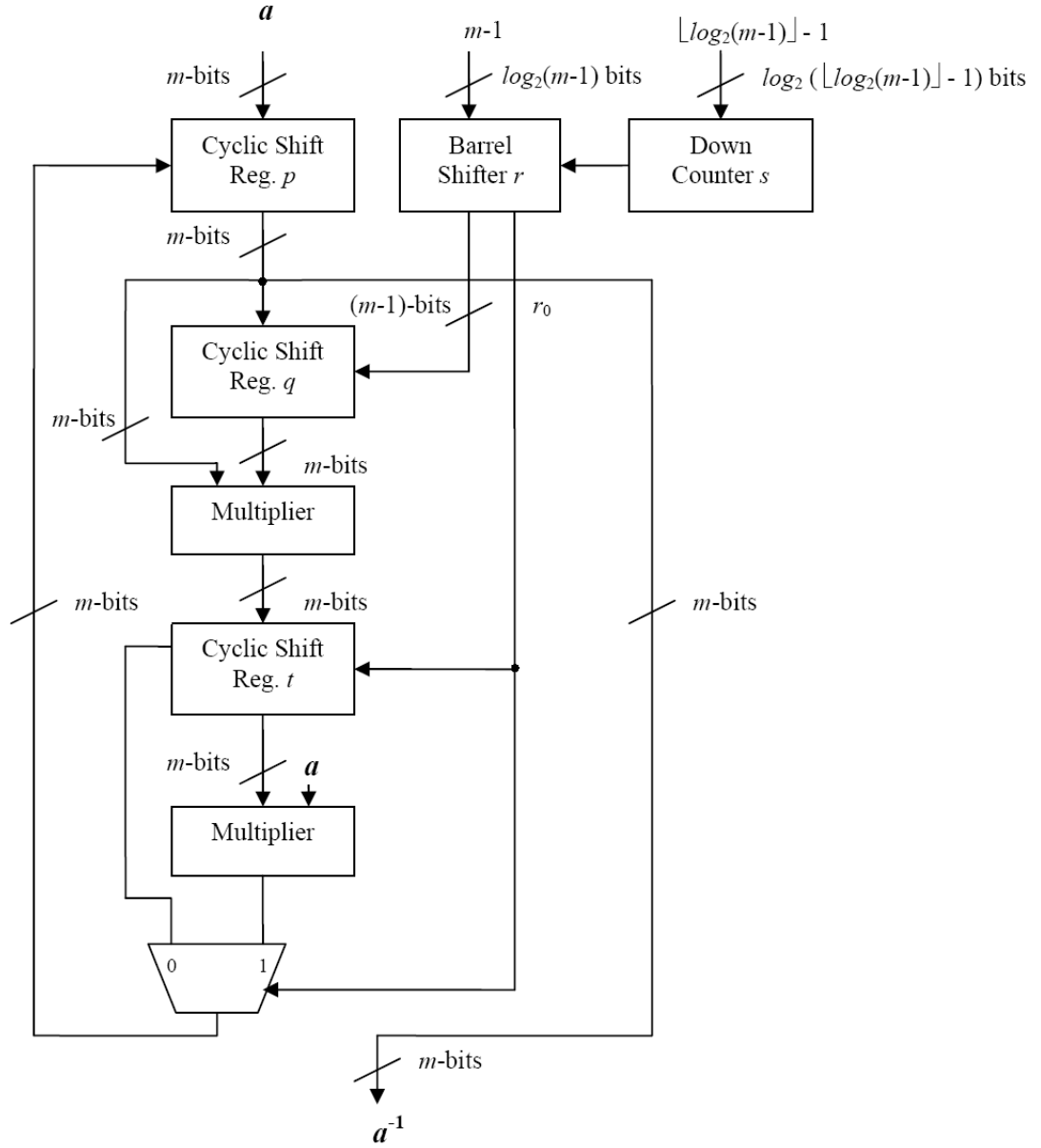


Figure 6.6: Dataflow of the Itoh and Tsujii inverter [87].

Algorithm 6.6 Itoh–Tsujii inversion algorithm.

Inputs: a

Output: $l = a^{-1}$

1. set $s \leftarrow \lfloor \log_2(m-1) \rfloor - 1$.
 2. set $p \leftarrow a$.
 3. for $i = s$ down to 0 do
 - 3.1. set $r \leftarrow$ shift $m-1$ to right by s bit(s)
 - 3.2. set $q \leftarrow p$
 - 3.3. rotate q to left by $\lfloor r/2 \rfloor$ bit(s)
 - 3.4. set $t \leftarrow p \times q$
 - 3.5. if least bit of $r = 1$,
 - 3.5.1 rotate t to left by 1 bit.
 - 3.5.2 $p \leftarrow t \times a$
 - 3.5. else
 - 3.5.3 $p \leftarrow t$
 - 3.6. $s \leftarrow s - 1$
 4. rotate p to left by 1 bit.
 5. set $l \leftarrow p$.
 6. return l .
-

Table 6.1: Lopez-Dahab Projective Coordinate System.

Point Doubling	Point Addition
$T_1 \leftarrow X_1$	$T_1 \leftarrow X_0$
$T_2 \leftarrow Y_1$	$T_2 \leftarrow Y_0$
$T_3 \leftarrow Z_1$	$T_3 \leftarrow Z_0$
$T_4 \leftarrow \sqrt{b}$	$T_4 \leftarrow X_1$
$T_3 \leftarrow T_3^2$	$T_5 \leftarrow Y_1$
$T_4 \leftarrow T_3 \times T_4$	$T_6 \leftarrow Z_1$
$T_4 \leftarrow T_4^2$	$T_7 \leftarrow T_3 \times T_6 = E$
$T_1 \leftarrow T_1^2$	$T_1 \leftarrow T_1 \times T_6 = B_1$
$T_3 \leftarrow T_1 \times T_3 = Z_2$	$T_4 \leftarrow T_3 \times T_4 = B_0$
$T_1 \leftarrow T_1^2$	$T_1 \leftarrow T_1 + T_4 = D$
$T_1 \leftarrow T_1 + T_4 = X_2$	$T_3 \leftarrow T_3^2$
$T_2 \leftarrow T_2^2$	$T_6 \leftarrow T_6^2$
if $a \neq 0$ then	$T_3 \leftarrow T_3 \times T_5 = A_0$
$T_5 \leftarrow a$	$T_6 \leftarrow T_2 \times T_6 = A_1$
$T_5 \leftarrow T_3 \times T_5$	$T_6 \leftarrow T_3 + T_6 = C$
$T_2 \leftarrow T_2 + T_5$	$T_2 \leftarrow T_1 \times T_7 = F$
$T_2 \leftarrow T_2 + T_4$	$T_1 \leftarrow T_1^2$
$T_2 \leftarrow T_1 \times T_2$	$T_8 \leftarrow T_7^2$
$T_4 \leftarrow T_3 \times T_4$	$T_8 \leftarrow a \times T_8$
$T_2 \leftarrow T_2 + T_4 = Y_2$	$T_8 \leftarrow T_2 + T_8$
	$T_5 \leftarrow T_1 \times T_8 = G$
	$T_8 \leftarrow T_2 \times T_6 = H$
	$T_6 \leftarrow T_6^2$
	$T_6 \leftarrow T_6 + T_8$
	$T_6 \leftarrow T_5 + T_6 = X_2$
	$T_4 \leftarrow T_1 \times T_4$
	$T_4 \leftarrow T_4 \times T_7$
	$T_4 \leftarrow T_4 + T_6 = I$
	$T_3 \leftarrow T_1 \times T_3$
	$T_3 \leftarrow T_3 + T_6 = J$
	$T_4 \leftarrow T_4 \times T_8$
	$T_2 \leftarrow T_2^2 = Z_2$
	$T_2 \leftarrow T_2 \times T_3$
	$T_8 \leftarrow T_3 + T_4 = Y_2$

6.2 The ECC_{SS} Cryptoprocessor

This section presents an original sequential elliptic curve cryptoprocessor which provides resistance against power analysis attacks at different levels. The private key is divided into a number of partitions that are processed independently. Security measures against power analysis attacks are provided at several levels: the key level, the key partition level and the individual bit level.

At the key level, the key is divided into a number of partitions which are sequentially processed in a randomized order. The points resulting from processing these key partitions are accumulated to produce the scalar product kP . Each key partition is associated with a precomputed point to keep its significance [73, 74]. The precomputed points are computed off-line and stored to be reused as needed. To increase the resistance against power analysis attacks, the key partitioning process, i.e. defining new key partition sizes and computing the corresponding values of the precomputed points, is performed from time to time.

Increasing the number of key partitions increases immunity against power analysis attacks since more key partitions provides more permutations. Increasing the number of key partitions, however, requires more storage for the precomputed points and more point additions to assimilate the partial computations into the final scalar product kP .

At the key partition level, two security countermeasures are adopted. The en-

coding of each key partition is randomly selected to be either in binary form or in Non-Adjacent-Form (NAF)[107]. Furthermore, at the key partition level, the direction of bit inspection for each key partition is randomly assigned to be either most-to-least or least-to-most if binary encoding is selected.

Finally, at the bit level, each zero in the key, may randomly perform a dummy point addition operation in addition to the doubling operation. Such zeros randomization increases the security and saves an average of 50% of the extra dummy point additions used in the double-and-add-always algorithm (Algorithm 5.1).

The multilevel protection scheme fully confuses any relation between the secret key and any leaked information resulting in a fairly secure system with minimal area and delay overhead. An attacker of such system will be totally confused with leaked information in such multilevel resistance secure environment.

6.2.1 Key Partitioning

The key is divided into u partitions as:

$$k = k^{(u-1)} || k^{(u-2)} || \dots || k^{(1)} || k^{(0)}$$

To compute the scalar product kP , these partitions are associated with a set of pre-computed points to keep the significance of each key partition, thus these partitions

can be processed independently either sequentially or in parallel.

$$\begin{aligned}
kP &= (k^{(u-1)} || k^{(u-2)} || \dots || k^{(1)} || k^{(0)}).P \\
&= (2^{\text{size}(u-1)} . k^{(u-1)} + 2^{\text{size}(u-2)} . k^{(u-2)} + \dots + 2^{\text{size}(1)} . k^{(1)} + k^{(0)}).P \\
&= (2^{\text{size}(u-1)} P).k^{(u-1)} + (2^{\text{size}(u-2)} P).k^{(u-2)} + \dots + (2^{\text{size}(1)} P).k^{(1)} + (P)k^{(0)} \\
&= P_{u-1}.k^{(u-1)} + P_{u-2}.k^{(u-2)} + \dots + P_1.k^{(1)} + P_0.k^{(0)} \\
&= P_{u-1}.k^{(u-1)} + P_{u-2}.k^{(u-2)} + \dots + P_1.k^{(1)} + P.k^{(0)}
\end{aligned}$$

where P_i ($i = 1, 2, \dots, u-1$) is the precomputed point associated with key partition $k^{(i)}$ and $\text{size}(j) = \left(\sum_{i=0}^{j-1} \text{size of key partition } k^{(i)} \right)$. Thus, each partition $k^{(i)}$ is associated with a precomputed point P_i forming the pair:

$$(k^{(i)}, P_i)$$

where $P_0 = P$.

The key partition sizes may be equal or different. For equal sizes, the key partition size is equal to $\lceil \frac{m}{u} \rceil$ for u key partitions. While equal sizes allow for simpler design, different size key partitions provides more security. In this work, different key sizes are used and the key partition sizes are randomly adjusted to avoid future attacks on equal key partition sizes.

Precomputed points are computed using a sequence of doubling operations of

the base point P . For u key partitions, the required number of precomputed points is $(u - 1)$. The resulting points of processing these key partitions are assimilated at the end to produce the scalar multiplication product $kP = \sum_{i=0}^{u-1} k^{(i)} P_i$ where $P_0 = P$.

A new set of precomputed points should be generated whenever the base point P or the number or sizes of key partitions are changed. In elliptic curve Diffie-Hellman protocol (Section 2.5.1), precomputations are performed off-line only once at the beginning since the base point of the two parties are not changed. Precomputations, however, should be performed whenever the number or sizes of key partitions are changed.

Alternatively, in elliptic curve ElGamal protocol (Section 2.5.2), the public point of the receiver is considered as the sender's base point. Accordingly, the sender uses this point together with his own key partitions to compute the required precomputed points once off-line. If another session is established between the two parties, new precomputed points need to be generated only if the sender changes his/her private key, the number of key partitions or the sizes of these partitions. The receiver, on the other hand, needs to generate a new set of precomputed points whenever the sender changes his/her private key or the number or sizes of the receiver key partitions are changed.

6.2.2 Multilevel Resistance Measures

To protect against power analysis attacks several resistance measures have been adopted to render the scalar multiplication process secure against these attacks. The private key consists of a group of bits with every bit having a particular position and a particular bit value. The adopted resistance measures depend on confusing not only the bit values but also the key bit positions. Thus, even if leaked information can identify the type of performed operation, e.g. point doubling or point addition, attackers can neither be sure of the corresponding key bit value nor its position.

Several resistance measures are proposed at different levels; the key level, the key partition level and the bit level. These resistance measures are described below.

Resistance Measures at The Key Level

The objective of the resistance measures at the key level is to confuse the key bit positions, thus leaked information cannot be associated with a known key bit position. The key is divided into u partitions which are sequentially processed in a randomized order to increase the resistance against power analysis attacks. The number of key partitions and their sizes are changed from time to time, which would require computing new associated precomputed points. Such computation is performed off-line.

Resistance Measures at The Key Partition Level

At this level, two resistance measures against power analysis attacks are proposed. First, to confuse the bit value, the encoding of each key partition is randomized to use either binary encoding or NAF encoding. In a NAF encoding, signed binary digit representation is used, i.e., each bit may be 0, 1 or $\bar{1}$. NAF has the property that no two consecutive bits are nonzero. Every integer has a unique NAF encoding. Moreover, NAF encoding has the fewest nonzero bits of any binary signed digit representation of an integer (Algorithm 6.7).

Second, to confuse the bit position, if binary encoding is selected in a particular key partition, the direction of bit inspection for this key partition is randomly assigned to be either most-to-least or least-to-most. This adds another level of resistance even if an attacker guessed correctly that a certain bit belongs to a certain key partition.

Algorithm 6.7 NAF encoding algorithm.

Inputs: A positive integer k .

Output: $\text{NAF}(k)$.

Initialization:

1. $i = 0$.
 2. While $k \geq 1$ do
 - 1.1. if $k \bmod 2 = 1$ then
 - 1.1.1. $k'_i = 2 - (k \bmod 2^2)$
 - 1.1.2. $k = k - k'_i$
 - 1.2. else
 - 1.2.1. $k'_i = 0$
 - 1.3. $k = k/2$
 - 1.4. $i = i + 1$
 3. Output (k')
-

Resistance Measures at The Bit Level

In the double-and-add-always algorithm (Algorithm 5.1), point doubling and addition are performed in each iteration regardless of the key bit value k_i . In Algorithm 5.1, the value of k_i is inspected such that:

- if $k_i = 1$, the results of doubling and addition are committed, otherwise
- if $k_i = 0$, only the result of doubling is committed while that of addition is ignored.

This simple approach caused the scalar multiplication to be resistant against SPA only. The drawback of this approach, however, is the delay overhead due to the extra dummy point additions and its vulnerability to DPA. In this work, another resistance technique is introduced at the bit level where a dummy point addition is randomly performed if $k_i = 0$. Thus, if the value of k_i is zero, a dummy point addition operation may or may not be performed based on the value of some random bit r as follows:

- if $r = 1$, perform the doubling operation together with a dummy addition operation.
- otherwise if $r = 0$, only the doubling operation is performed.

The most-to-least version of the proposed randomized bit algorithm is given in Algorithm 6.8. In Algorithm 6.8, point doubling is always performed in Step

2.1. Point additions are performed according to the value of k_i and the random bit r at Step 2.2.1. Similarly, Algorithm 6.9 shows the least-to-most version of the randomized bit algorithm. In Algorithm 6.9, point addition is performed according to the value of k_i and the random bit r at Step 3.1.1 while point doubling is always performed at Step 3.2.

Algorithm 6.8 The Randomized bit Algorithm (most-to-least)

Inputs: P :A precomputed point, k :A key partition, k_{size} .

Output: kP : Partial scalar product.

Initialization:

1. $Q[0] = P \cdot k_{(k_{size}-1)}$

Scalar Multiplication:

2. for i from $k_{size} - 2$ to 0 do

2.1. $Q[0] = 2Q[0]$

2.2. if $(k_i = 1$ or $r = 1)$ then

2.2.1. $Q[1] = Q[0] + P$

2.3. $Q[0] = Q[k_i]$

3. Output ($Q[0]$)

Algorithm 6.9 The Randomized bit Algorithm (least-to-most)

Inputs: P :A precomputed point, k :A key partition, k_{size} .

Output: kP : Partial scalar product.

Initialization:

1. $Q[0] = P$

2. $Q[1] = \mathcal{O}$

3. for i from 0 to $k_{size} - 1$ do

3.1. if $(k_i = 1$ or $r = 1)$ then

3.1.1. $Q[2] = Q[1] + Q[0]$

3.2. $Q[0] = 2Q[0]$

3.3. $Q[1] = Q[1 + k_i]$

4. Output ($Q[1]$)

Example

Figure 6.7 shows an example of key partitioning and execution scheduling. In the example of Figure 6.7, the key length is 16-bit. The first step is to partition the key k into a number of partitions (4 in this example), i.e.,

$$k = k^{(3)} \| k^{(2)} \| k^{(1)} \| k^{(0)} \quad (6.8)$$

The second step is to randomly arrange the key partitions to form the new randomized key.

$$k_{new} = k^{(2)} \| k^{(0)} \| k^{(1)} \| k^{(3)} \quad (6.9)$$

The third step is to randomly encode each key partition either in binary or NAF representation and randomly assign the direction of inspection of key partition bits (most-to-least or least-to-most) if binary encoding is selected. Note that key partition $k^{(3)}$ is encoded in NAF representation. Finally, the fourth step shows how the randomized zeros algorithm behaves.

Security, Space and Time Analysis

Although an attacker may be able to distinguish the double and add point operations, the adopted multi-level resistance measures do not allow the attacker to associate this operation with a specific key bit position on one hand, nor to ascertain a particular binary bit value to it on the other.

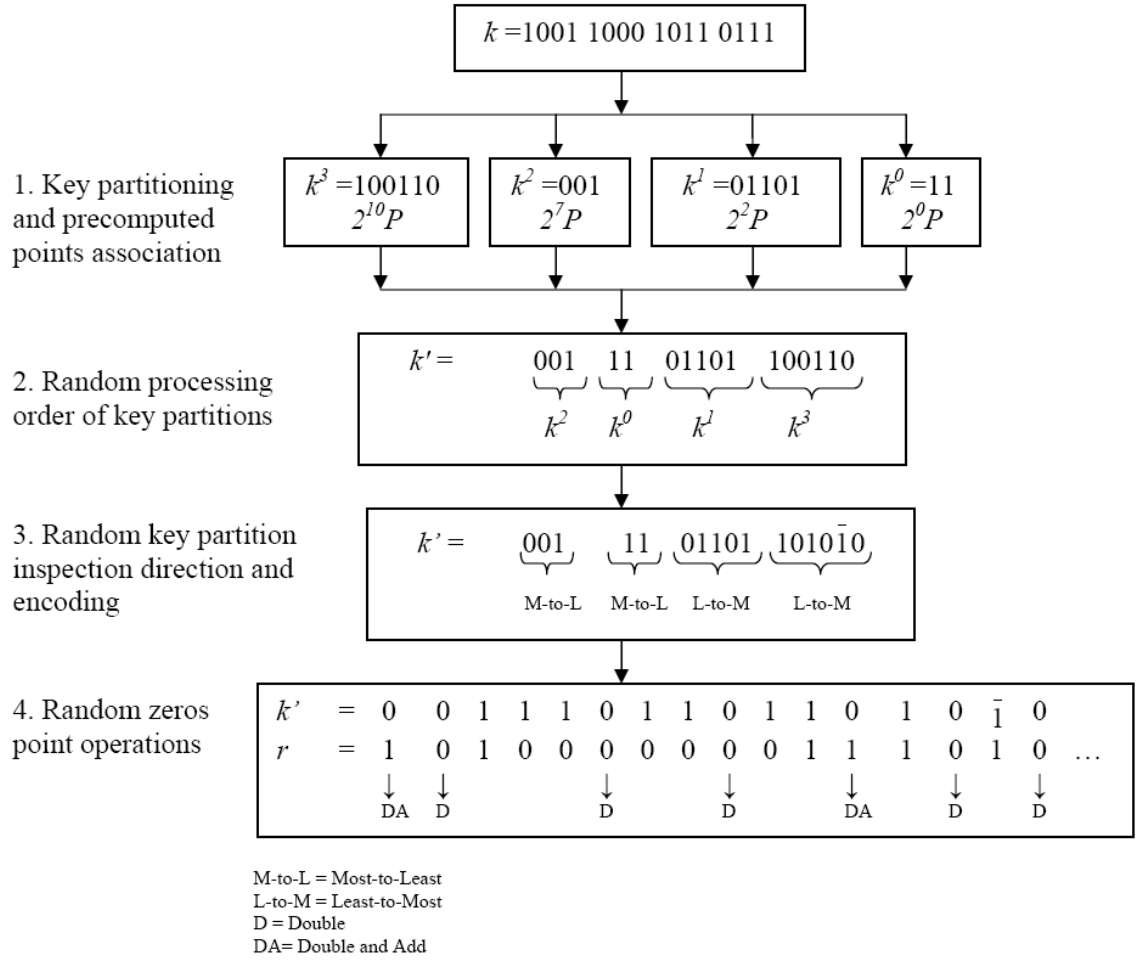


Figure 6.7: Multilevel resistance measures.

The multi-level resistance measures will confuse prospective attackers regarding the exact bit positions of the key since:

1. Key partitions are processed in a randomized order.
2. The inspection direction of key partitions that are encoded in binary is randomized (either most-to-least or least-to-most).
3. Key partitions that are NAF encoded may have a size that is greater by 1 bit than its corresponding binary encoded partition size.

Furthermore, key bit values cannot be definitely ascertained by prospective attackers. Even if a given bit appears to an attacker as if it is a binary 1 its true value cannot be ascertained since this bit may (1) have a true binary 1 value, (2) have a true binary 0 value with a dummy add operation or (3) have a $\bar{1}$ value in case of NAF encoding. Likewise, a recognized zero-bit, may be (1) a true zero in the binary form or (2) a zero in a NAF encoding.

Resistance against the double attack [96] is achieved because double attack targets the most-to-least version of the double-and-add algorithm. The proposed cryptoprocessor is designed to perform both versions of the double-and-add algorithm even at the key partition level.

Furthermore, the proposed sequential architecture is also secure against RPA and ZPA. To protect against RPA and ZPA, either the base point P or the secret scalar k should be randomized. The proposed key partitioning with random processing

order scheme, random partition encoding, random partition bits inspection direction and random dummy additions makes the secret scalar k appear as if it is totally randomized.

The proposed multilevel randomization techniques make it very difficult to establish a correlation between the secret information and addresses of registers. Thus, the proposed cryptoprocessor is secure against ADPA.

Even though increasing the number of key partitions provides more security, an increased number of key partitions (u) results in more space overhead since the number of precomputed points ($u - 1$) will increase accordingly. Likewise, the delay overhead will also increase by increasing the number of key partitions since ($u - 1$) extra point additions are required to assimilate the partial results to produce the scalar product kP .

NAF encoding requires, on the average, $\frac{m}{3}$ point additions and hence provides better time performance than binary encoding which requires, on the average, $\frac{m}{2}$ point additions. NAF encoding, however, requires signed bit representation and may increase the size of key partition by at most 1 bit.

The inspection direction does not cause any delay overhead since the time required to perform scalar multiplication using Algorithm 6.8 is the same time required by Algorithm 6.9 which requires one more point storage than what Algorithm 6.8 requires.

While dummy computations caused by the randomized bit algorithms (Algo-

rithms 6.8 and 6.9) increases the degree of confusion, it does increase the number of point additions. Although this may significantly increase the time overhead, it is still a more attractive approach compared to the double-and-add-always algorithm (Algorithm 5.1).

6.2.3 ECC_{SS} Architecture and Operation

Both of the ECC_{NS} and the ECC_{SS} cryptoprocessors use the same basic blocks which include: (1) the point addition and doubling units, (2) the field arithmetic units (multiplier and inverter) and (3) the data embedding unit. The two cryptoprocessors differ only in the control path and the number of extra registers used by the the ECC_{SS} cryptoprocessor to store the precomputed points and the accumulation point.

The pseudocode of ECC_{SS} is given in Algorithm 6.10. Algorithm 6.10 uses the same inputs/outputs as Algorithm 6.1. Key partitioning, precomputations, precomputed point association with key partitions are assumed to be performed off-line. The key is partitioned into u partitions. Sizes of key partitions may be equal or different. Different key partition sizes provides more security. Thus, the size of key partitions are randomly adjusted to avoid future attacks on equal key partition sizes. If NAF encoding is selected, the key partition size may be increased by up to one bit since NAF encoding requires at most one extra bit.

Precomputations are performed by repeated double operations in Steps 1-3 ac-

cording to key partition sizes. The number of required precomputed points are $(u - 1)$. Each key partition $k^{(i)}$ is associated with a particular precomputed point P_i to keep the significance of each key partition (Step 4).

Scalar multiplication starts at Step 5 after random arrangement of key partitions. The inspection direction is randomly selected if a key partition is binary encoded. Steps 5.1-5.1.1 show the steps if the inspection direction is most-to-least with binary encoding. If NAF encoding is used, each bit of the scalar multiplier $k'^{(i)}$ is recoded in NAF representation and accordingly point operations are performed (Steps 5.2.1-5.2.3). Each partition is processed as if it is a key itself. The partial points resulting from processing the individualized key partitions are accumulated in the point R (Step 5.3) which requires $(u - 1)$ extra point additions.

6.3 The ECC_{PS} Cryptoprocessor

This section describes our proposed high performance parallel elliptic curve cryptoprocessor with resistance against power analysis attacks. The main idea is to partition the secret key into a number of partitions, as described in Section 6.2.1, that are processed in parallel by independent scalar multiplication units. Key partitions inspection direction (most-to-least or least-to-most) are independently selected for each scalar multiplication unit in a randomized manner.

Parallel inspection of the key bits cause point operations being executed by the

Algorithm 6.10 Pseudocode of the ECC_{SS} cryptoprocessor.

Inputs: P : Base Point, k : Secret key, a , b : Elliptic curve parameters,
Plaintext/Ciphertext, Encryption/Decryption.

Outputs: Ciphertext/Plaintext.

Key Partitioning: $k = k^{(u-1)} || k^{(u-2)} || \dots || k^{(1)} || k^{(0)}$, for u key partitions.

Initialization: $Q = P$, $R = \mathcal{O}$.

Precomputation:

1. $P_0 = Q$.
2. for $i = 1$ to $u - 1$ do
 - 2.1. for $j = 0$ to $k_{size}^{(i-1)} - 1$ do
 - 2.1.1 $Q = 2Q$
 - 2.2. end for
 - 2.3. $P_i = Q$
3. end for

Key Partitions Association with Precomputed Points:

4. for $i = 0$ to $u - 1$ do $(k'^{(i)}, P_i)$.

Key after random rearrangement: $k' = k'^{(u-1)} || k'^{(u-2)} || \dots || k'^{(1)} || k'^{(0)}$.

Scalar Multiplication ($R = kP$):

5. for $i = 0$ to $u - 1$ do
 - 5.1 if (Binary) then
 - 5.1.1. if (most-to-least) then $Q = \text{Algorithm 6.8 } (k'^{(i)}, P_i, k_{size}^{(i)})$
 - 5.1.2. else $Q = \text{Algorithm 6.9 } (k'^{(i)}, P_i, k_{size}^{(i)})$
 - 5.2. else (NAF)
 - 5.2.1. $j = 0$, $t = k'^{(i)}$, $Q[0] = P$, $Q[1] = \mathcal{O}$
 - 5.2.2. While $t \geq 1$ do
 - 5.2.2.1. if $t \bmod 2 = 1$ then
 - 5.2.2.1.1. $k_j^{(i)} = 2 - (t \bmod 2^2)$
 - 5.2.2.1.2. $t = t - k_j^{(i)}$
 - 5.2.2.2. else
 - 5.2.2.2.1. $k_j^{(i)} = 0$
 - 5.2.2.3. $t = t/2$, $j = j + 1$
 - 5.2.2.4. if $(k_j^{(i)} = 1 \text{ or } r = 1)$ then $Q[2] = Q[1] + Q[0]$
 - 5.2.2.5. else if $(k_j^{(i)} = -1 \text{ or } r = 1)$ then $Q[2] = Q[1] - Q[0]$
 - 5.2.2.6. $Q[0] = 2Q[0]$
 - 5.2.2.7. $Q[1] = Q[1 + |k_j^{(i)}|]$
 - 5.2.3. $Q = Q[1]$
 - 5.3. $R = R + Q$
6. end for

Encryption/Decryption Process: Same as in Algorithm 6.1.

scalar multiplication units to overlap at various stages of point and individual field operations. Thus, multiple field operations are fused together causing the power trace of these simultaneous operations to confuse the nature of point operations being performed by the parallel scalar multiplication units. Furthermore, the randomized inspection order of each key partition increases immunity against power analysis attacks. This adds another layer of security and confuses attackers monitoring power trace.

The number of key partitions is limited by the number of available scalar multiplication units. Each key partition is associated with a precomputed point to keep its significance. Precomputed points are computed off-line and stored to be reused as needed. For u key partitions, $(u - 1)$ precomputed points are required. The selection of these points depends on the number and sizes of the key partitions. Each partition is associated with a precomputed point P_i as:

$$(k^{(i)}, P_i) \tag{6.10}$$

where $P_0 = P$.

The sizes of key partitions may be equal or different. For u key partitions, u scalar multiplication units are required since each key partition requires an individual scalar multiplication unit. An extra scalar multiplication unit is also required to accumulate the partial computations into the final scalar product kP . Different key

partition sizes allows for optimizing the required number of scalar multiplication units. The scalar multiplication unit with the smallest key partition size can be utilized to accumulate the partial computations into the final scalar product kP . This saves the need for extra scalar multiplication unit which is needed for points accumulation.

Using different key partition sizes also increases the immunity against power analysis attacks and future attacks on fixed key partition size. Thus, different key partitions sizes are used in this work. To increase the resistance against power analysis attacks more, the key partitioning process and accordingly precomputation process of the required precomputed points are performed from time to time.

The ECC_{PS} cryptoprocessor does not need extra dummy operations as performed in the ECC_{SS} cryptoprocessor. Hence, high performance is achieved while, at the same time, improving the resistance against power analysis attacks. Leaked information, in this case, are quite confusing since it will be a combination of different point operations being performed at the same time. The proposed cryptoprocessor requires several scalar multipliers to perform scalar multiplications in parallel as depicted in Figure 6.8.

Example

Figure 6.9 shows an example of the parallel execution of key partitions. In the example of Figure 6.9, the key length is 16-bits. The first step is to precompute

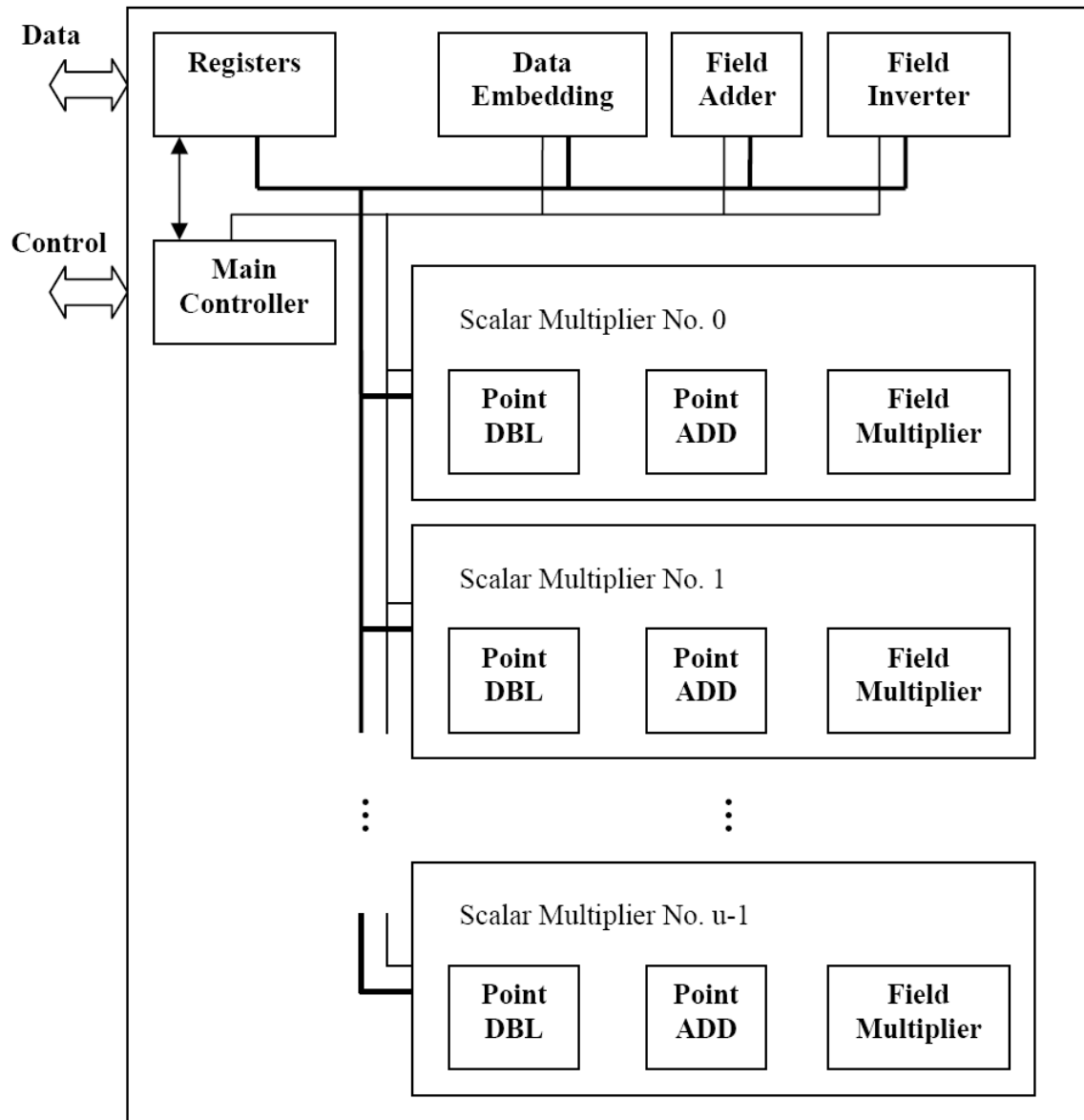


Figure 6.8: The proposed parallel architecture.

several points and store them in registers. Each of these precomputed points is associated with exactly one key partition.

The second step is to partition the key k into u key partitions ($u = 4$ in this example). Thus,

$$k = k^{(3)} \| k^{(2)} \| k^{(1)} \| k^{(0)} \quad (6.11)$$

The third step is to randomly assign the direction of inspection of key partition bits (most-to-least or least-to-most). The fourth step is the parallel execution of scalar multiplications as illustrated in Figure 6.9. In Lopez–Dahab projective coordinate system, point addition requires mainly 14 multiplications while point doubling requires only 5 multiplications (Table 6.1). Thus, point addition requires around three times as many multiplications required by point doubling. Accordingly, leaked information at a certain point in time will be a combination of field operations of point doubling, additions and no operations (Figure 6.9). This makes it very difficult to infer the key from any leaked information. Finally, the resulting points of all partition are assimilated to produce the final scalar product kP (note that an extra scalar multiplication unit is used in Figure 6.9 to accumulate to partial products for the purpose of clarity).

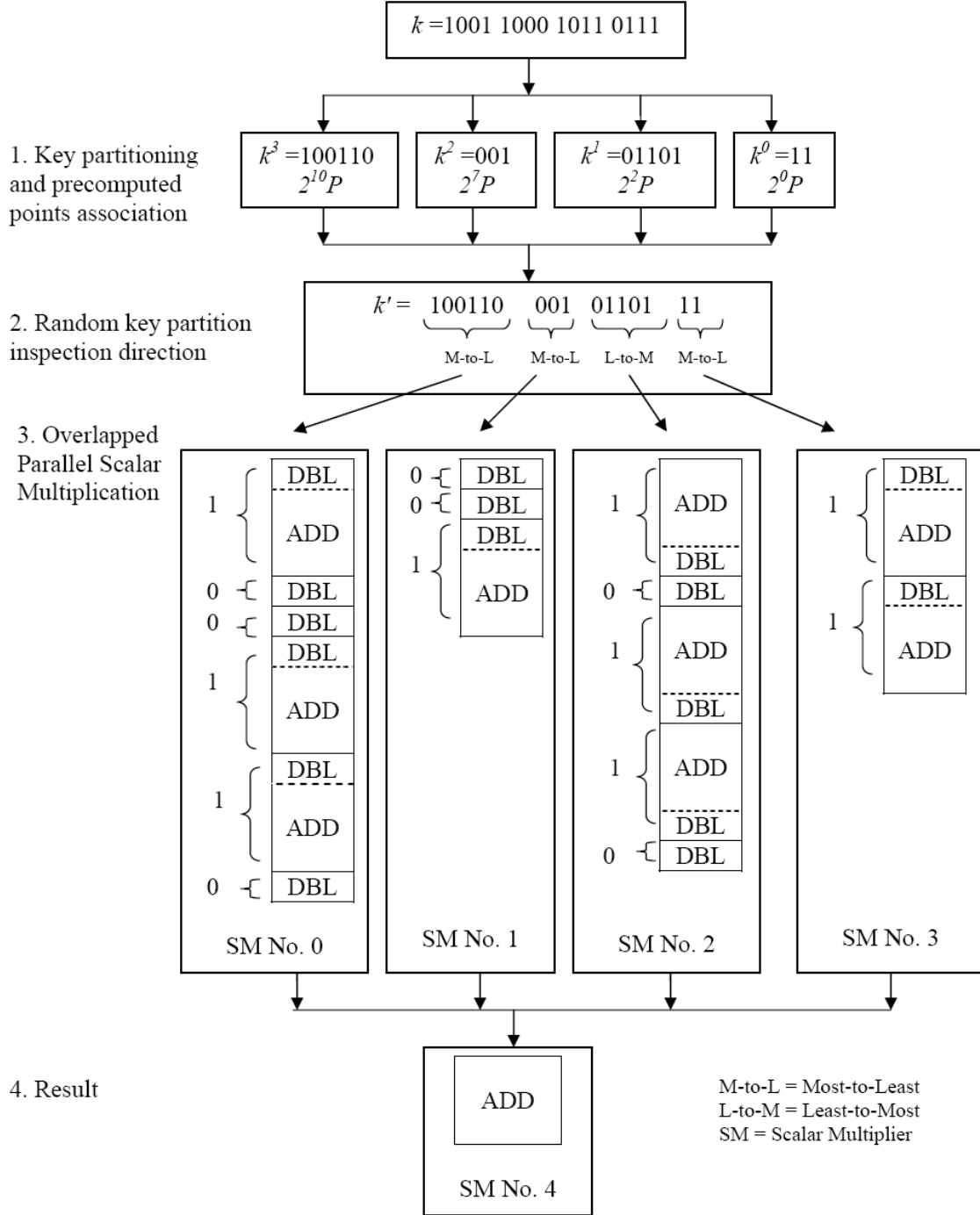


Figure 6.9: Key partitioning and parallel execution.

6.3.1 ECC_{PS} Architecture and Operation

As shown in Figure 6.8, the architecture of the ECC_{PS} cryptoprocessor is different from the architecture of the ECC_{SS} cryptoprocessor. The ECC_{PS} cryptoprocessor uses parallel scalar multiplication units. The number of these scalar multiplication units depends on the number of the key partitions. Each scalar multiplier consists of: (1) a field multiplier, (2) a point addition unit, and (3) a point doubling unit. The field arithmetic units (adder and inverter) and the data embedding unit are identical to those used in the ECC_{NS} and the ECC_{SS} cryptoprocessors.

The pseudocode of ECC_{PS} is given in Algorithm 6.11. Key partitioning, pre-computations, precomputed point association with key partitions are assumed to be performed off-line. The key is partitioned into u partitions. Different key partition sizes are used and the sizes of these key partitions are adjusted randomly. Precomputations are performed by repeated doubles in Steps 1-3 according to sizes of key partitions and the number of required precomputed points are $(u - 1)$. Each key partition $k^{(i)}$ is associated with a particular precomputed point P_i to keep the significance of each key partition (Steps 4-5).

Parallel scalar multiplications start at Step 6. The inspection direction of each key partition is randomly selected. Modified versions of the double-and-add algorithms (Algorithms 6.12 and 6.13) are proposed here to be used by each scalar multiplier since key partitions have different sizes. Each partition is processed as

if it is a key itself. The resulting points of each execution of Algorithms 6.8 and 6.9 are accumulated in the point R (Step 6.3) which requires $(u - 1)$ extra point additions. The time required to perform these extra additions is not significant since the accumulation process is performed as soon as a point is produced by a scalar multiplication unit. Finally, the encryption/decryption process is exactly the same as performed in the ECC_{NS} cryptoprocessor.

6.3.2 Security, Space and Time Analysis

In addition to high speed operation, the proposed parallel architecture provides a built-in countermeasure against power analysis attacks which ensures that attackers can not distinguish between point operations or their boundaries. The overlapped parallel point and field operations and random key partition inspection order confuse attackers about which bits of which partitions of the key are being processed.

Increasing the number of key partitions increases the immunity against power analysis attacks as well as the speedup factor. The required space, however, increases significantly as more key partitions are used since each key partition requires an individual scalar multiplier. Accordingly, there is tradeoff between speed and security on one hand and area on the other with different resistance strength against power analysis attacks.

The double attacks [96] cannot be applied to the ECC_{PS} cryptoprocessor because double attacks only target the most-to-least version of the sequential double-and-

add algorithm. The proposed cryptoprocessor architecture performs parallel scalar multiplications and implements both versions of the double-and-add algorithm randomly.

Furthermore, the proposed cryptoprocessor is also secure against RPA and ZPA since leaked information about the key are both overlapped and randomized. It is very difficult for an attacker to infer the secret key.

Finally, correlations between the secret information and addresses of registers are extremely hard to be analyzed since parallelism and randomization are employed together. Thus, the proposed cryptoprocessor is also secured against ADPA.

6.4 Summary

In this chapter, three elliptic curve cryptoprocessor architectures for curves defined over $GF(2^m)$ have been proposed. Two of these architectures are designed to be secure against power analysis attacks, while the third non-secure one is designed to be used as a reference model for area and delay comparisons.

The proposed power analysis attack resistant cryptoprocessors include one sequential and another parallel cryptoprocessor architectures. The sequential cryptoprocessor architecture uses multilevel resistance measures against power analysis attacks. These include resistance measures at the key level, the key partition level, and at the bit level.

At the key level, the key is divided into a number of partitions which are sequentially processed in a randomized order. The encoding of each key partition is randomly selected to be either in binary or in Non-Adjacent-Form (NAF) at the key partition level. Furthermore, at the key partition level, the direction of bit inspection for binary encoded key partitions is randomly assigned to be either most-to-least or least-to-most. Finally, the zeros, at the bit level, are randomized to appear sometimes as ones by performing dummy point additions.

The proposed parallel cryptoprocessor provides high speed through using parallel scalar multipliers that operate in parallel on different key partitions. Parallel scalar multiplications of different key partitions are exploited as a countermeasure against power analysis attacks. Furthermore, the inspection order of each key partition is randomly decided to increase the immunity against power analysis attacks. The parallel cryptoprocessor does not need extra dummy operations as performed in the sequential processor.

Algorithm 6.11 Pseudocode of the ECC_{PS} cryptoprocessor.

Inputs: P : Base Point, k : Secret key, a , b : Elliptic curve parameters,
Plaintext/Ciphertext, Encryption/Decryption.

Outputs: Ciphertext/Plaintext.

Key Partitioning:

$k = k^{(u-1)} || k^{(u-2)} || \dots || k^{(1)} || k^{(0)}$, where u is the number of key partitions.

Initialization:

$Q = P$, $R = \mathcal{O}$.

Precomputation:

1. $P_0 = Q$.
2. for $i = 1$ to $u - 1$ do
 - 2.1. for $j = 0$ to $k_{size}^{(i-1)} - 1$ do
 - 2.1.1 $Q = 2Q$
 - 2.2. end for
 - 2.3. $P_i = Q$
3. end for

Key Partitions Association with Precomputed Points:

4. for $i = 0$ to $u - 1$ do
 - 4.1. $(k^{(i)}, P_i)$.
5. end for

Parallel Scalar Multiplication ($R = kP$):

6. for $i = 0$ to $u - 1$ do in Parallel
 - 6.1 if (most-to-least) then
 - 6.1.1. $Q = \text{Algorithm 6.12 } (k^{(i)}, P_i, k_{size}^{(i)})$
 - 6.2 else
 - 6.2.1. $Q = \text{Algorithm 6.13 } (k^{(i)}, P_i, k_{size}^{(i)})$
 - 6.3. $R = R + Q$.
7. end for

Encryption/Decryption Process: Same as in Algorithm 6.1.

Algorithm 6.12 Modified double-and-add scalar multiplication algorithm (most-to-least).

Inputs: P : A precomputed point, k : A key partition, k_{size} .

Output: kP : Partial scalar product.

Initialization:

1. $Q = P \cdot k_{(k_{size}-1)}$

Scalar Multiplication:

2. for $i = k_{size} - 2$ down to 0 do

2.1. $Q = 2Q$

2.2. if $k_i = 1$ then $Q = Q + P$

end for

3. return(Q)

Algorithm 6.13 Modified double-and-add scalar multiplication algorithm (least-to-most).

Inputs: P : A precomputed point, k : A key partition, k_{size} .

Output: kP : Partial scalar product.

Initialization:

1. $Q = \mathcal{O}$, $R = P$

Scalar Multiplication:

2. for $i = 0$ to $k_{size} - 1$ do

2.1. if $k_i = 1$ then $Q = Q + R$

2.2. $R = 2R$

end for

3. return(Q)

Chapter 7

Results and Discussions

To evaluate our proposed secure ECC architectures (ECC_{SS} and ECC_{PS}) against the reference non-secure architecture (ECC_{NS}), the three architectures were modeled using VHDL and synthesized on Xilinx FPGA. The developed VHDL models are parameterized to allow synthesizing the cryptoprocessors with different architectural features. The developed VHDL allow for flexible definition of the following parameters:

1. The elliptic curve parameters a and b .
2. The underlying field $GF(2^m)$.
3. The base point P .
4. The secret key k .
5. The number of key partitions for the secure ECC_{SS} cryptoprocessor.

6. The number of parallel scalar multiplication units for the secure ECC_{PS} cryptoprocessor.

This chapter presents the results of synthesizing the various cryptoprocessors and compares these three cryptoprocessors in terms of delay and area. Xilinx (xc2v8000) FPGA has been used for prototyping. The reason for selecting such high capacity FPGA is to use the same FPGA chip with the three cryptoprocessors. This is essential to ensure that delay and area comparisons are done for the same technology and FPGA architecture and resources. The voltage trace of the ECC_{PS} cryptoprocessor is also illustrated and discussed in this chapter.

7.1 The ECC_{NS} Cryptoprocessor

The three cryptoprocessors were designed to use the same field operation algorithms, e.g., multiplication and inversion. Thus, performance difference between these cryptoprocessors is mainly a function of their control strategy and architectural differences independent of field operations. For example, field multiplication requires $(m + 2)$ clock cycles because of the sequential version of Sunar and Koc multiplier (Section 6.1). Two clock cycles are required for conversion from and back to optimal normal basis. Multiplication in the shifted basis requires m clock cycles. Point doubling requires 5 field multiplications, 4 field additions and 6 squarings. Each field addition and squaring requires only one clock cycle as a result of using optimal nor-

mal basis. The total number of clock cycles required for performing point doubling is $5(m + 2) + 10$, i.e., $(5m + 20)$ clock cycles.

Point addition, on the other hand, requires 14 field multiplications, 8 field additions and 6 squarings which requires $14(m + 2) + 14$, i.e. $(14m + 42)$ clock cycles. Scalar multiplication requires, on the average, m point doubles and $\lfloor \frac{m}{2} \rfloor$ point additions, using the double-and-add algorithm. Thus, the required time to perform scalar multiplication, on the average, is $m(5m + 20) + \lfloor \frac{m}{2} \rfloor(14m + 42)$, i.e. $(12m^2 + 41m)$ clock cycles.

The ECC_{NS} cryptoprocessor has been synthesized on the Xilinx FPGA (xc2v8000) which contains 46592 Slices. Table 7.1 shows the synthesis results of scalar multiplication with $m = 14, 30, 65, 90$ and 173 bits. As expected, these results show a linear increase in area.

Table 7.1: The ECC_{NS} Cryptoprocessor Synthesis Results.

m	DBLs	Adds	Clock(MHz)	Delay(μ sec)	Area (Slices)	Area Usage
14	14	7	93.954	31.14	1602	3%
30	30	15	74.235	162.05	3380	7%
65	65	32	64.595	826.15	8445	18%
90	90	45	60.055	1679.96	11933	25%
173	173	86	53.454	6851.52	27504	59%

7.2 The ECC_{SS} Cryptoprocessor

The ECC_{SS} cryptoprocessor, presented in Chapter 6, has also been synthesized on the same Xilinx FPGA (xc2v8000) with $m = 14, 30, 65, 90$ and 173 bits. For the ECC_{SS} cryptoprocessor, the number of point doubles remains the same as with the ECC_{NS} cryptoprocessor which is equal to m . The ECC_{SS} cryptoprocessor randomly encodes each key partition either in binary or NAF encoding.

Binary encoding requires, on the average, $\frac{m}{2}$ point additions, while NAF encoding requires, on the average, only $\frac{m}{3}$ point additions. Thus, an average of $\frac{5}{12}m$ point additions will be performed due to the 1's in the key partitions encoding (binary & NAF). At the bit level, in the ECC_{SS} cryptoprocessor, zeros of the key partitions may randomly cause dummy point additions. Since key partitions may be encoded either in binary or in NAF representation, an average of $\frac{7}{24}m$ dummy point additions are also required. Thus, the total number of point additions, on the average, is $\frac{17}{24}m \simeq 0.7m$.

The accumulation process of u partitions requires $(u - 1)$ extra point additions. Thus, the average total number of point additions required by the ECC_{SS} cryptoprocessor is approximately $(0.7m + u - 1)$. Table 7.2 shows the synthesis result for $u = 2, 3$ and 4 key partitions for $m = 14, 30, 65, 90$ and 173 bits. Clearly, the required time to perform scalar multiplication increases by increasing the number of key partitions u since $(u - 1)$ extra point additions are required for points ac-

cumulation. The space, in terms of required number of slices on the Xilinx FPGA (xc2v8000), also increases by increasing the number of key partitions u because of the need to store more precomputed points.

To evaluate the performance of the secure cryptoprocessors ECC_{SS} and ECC_{PS} , we will use the normalized area overhead and normalized delay overhead as measures of performance.

The delay and area overhead of both processors are measured by the normalized delay and area of both processors with respect to the sequential non-secure reference cryptoprocessor ECC_{NS} . Accordingly, the delay and area overheads are defined as:

$$\text{Delay Overhead} = \frac{ECC_{XS} \text{ Delay}}{ECC_{NS} \text{ Delay}}$$

and

$$\text{Area Overhead} = \frac{ECC_{XS} \text{ Area}}{ECC_{NS} \text{ Area}}$$

where ECC_{XS} represents either the ECC_{SS} or ECC_{PS} cryptoprocessor.

For the ECC_{SS} , Figure 7.1 shows that increasing the size of the underlying field $GF(2^m)$ decreases the delay overhead since the number of required extra point additions is only $(u - 1)$ for $u = 2, 3$ and 4 . Thus, if u is selected to be very large, the delay overhead is going to increase significantly.

Likewise, the area overhead decreases as m is increased since the common modules are not changed. Figure 7.1 also shows that for $m \geq 65$, increasing u does not significantly increase the area overhead. Accordingly, more security against power

analysis attacks can be attained by increasing u at the expense of more area and delay overhead.

Table 7.2: The ECC_{SS} Cryptoprocessor Synthesis Results.

m	u	DBLs	Adds	Clock (MHz)	Delay (μ sec)	Delay Overhead	Area (Slices)	Area Overhead
14	2	14	11	93.954	40.77	1.31	1873	1.169
14	3	14	12	93.954	43.30	1.39	1973	1.232
14	4	14	13	93.954	45.83	1.47	2002	1.249
30	2	30	22	74.172	205.79	1.27	3934	1.164
30	3	30	23	74.108	212.20	1.31	4110	1.216
30	4	30	24	74.108	218.44	1.35	4228	1.251
65	2	65	47	64.295	1037.29	1.25	9696	1.148
65	3	65	48	64.295	1052.10	1.27	10091	1.195
65	4	65	49	64.295	1066.91	1.29	10278	1.217
90	2	90	64	60.055	2091.88	1.24	13496	1.128
90	3	90	65	60.055	2113.56	1.26	14015	1.174
90	4	90	66	60.055	2135.24	1.27	14323	1.200
173	2	173	122	53.454	8492.52	1.24	31053	1.129
173	3	173	123	53.454	8538.62	1.25	31660	1.151
173	4	173	124	53.114	8639.67	1.26	32757	1.191

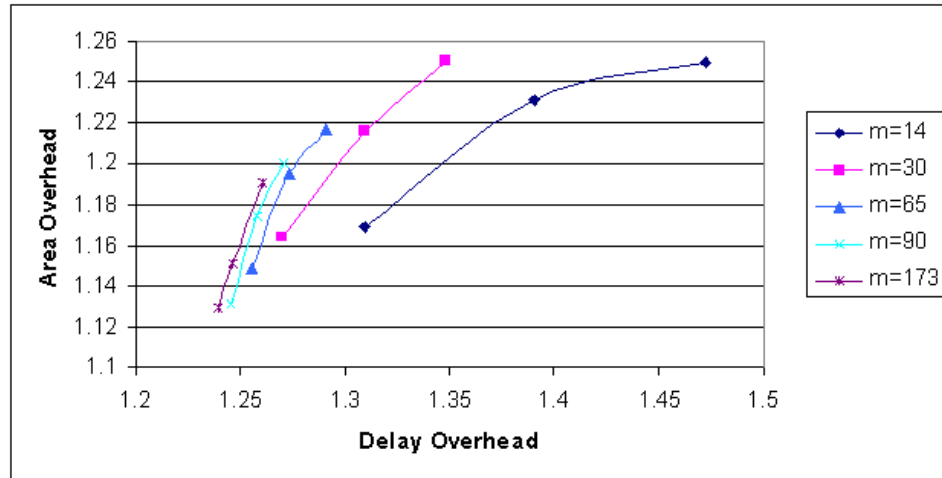


Figure 7.1: The ECC_{SS} Cryptoprocessor Delay and Area Overheads.

7.3 The ECC_{PS} Cryptoprocessor

As expected, since the ECC_{PS} cryptoprocessor uses u parallel scalar multipliers, it requires more hardware resources. Thus, for the Xilinx FPGA (xc2v8000), the synthesis report shows that the ECC_{PS} cryptoprocessor cannot fit with two or more parallel scalar multiplication units with $m = 173$ bits. Thus, only $GF(2^m)$ fields of $m = 14, 30, 65$ and 90 bits have been synthesized with up to four parallel scalar multipliers.

Table 7.3 shows synthesis results of the ECC_{PS} cryptoprocessor. Obviously, increasing the number of key partitions u decreases the execution time of scalar multiplication since all key partitions are processed in parallel. The required number of FPGA slices, on the other hand, increases significantly by increasing u since the number of scalar multiplication units also equals u .

Figure 7.2 shows the normalized delay and area overheads of the ECC_{PS} cryptoprocessor with respect to the ECC_{NS} cryptoprocessor. The efficiency, in terms of delay and area overheads, clearly increases as the number of bits in the underlying field $GF(2^m)$ increases.

The ECC_{PS} cryptoprocessor has been implemented on Xilinx (xcv300) FPGA with $m = 11$ -bits on XESS XSV V1.1 board. In order to measure the trace of the core logic power supply, a $1.2\ \Omega$ resistor has been placed in series with the core logic power supply. Figure 7.3 shows the voltage ranges of the ECC_{PS} cryptoprocessor

with up to three parallel scalar multipliers. If only a single scalar multiplier is used, point addition and point doubling can be easily distinguished, while with two scalar multipliers, the voltage ranges are overlapped as shown in Figure 7.3.

Further, using three parallel scalar multipliers makes it very difficult to distinguish between different cases that may occur such as 2 doubling + 1 addition or 2 additions + 1 doubling. Parallel scalar multiplications also overlap field operations across different scalar multipliers. The results depicted in Figure 7.3 clearly show that parallelism is an effective countermeasure.

Table 7.3: The ECC_{PS} Cryptoprocessor Synthesis Results.

m	u	DBLs	Adds	Clock (MHz)	Delay (μ sec)	Delay Overhead	Area (Slices)	Area Overhead
14	2	14	8	94.413	16.76	0.538	2902	1.812
14	3	14	9	94.311	12.02	0.386	4264	2.662
14	4	14	10	93.099	9.77	0.314	4837	3.0193
30	2	30	16	73.982	84.43	0.521	6091	1.802
30	3	30	17	73.700	58.59	0.362	8865	2.623
30	4	30	18	73.607	45.57	0.281	10089	2.985
65	2	65	33	63.127	426.96	0.521	14365	1.701
65	3	65	34	63.127	289.67	0.353	20423	2.418
65	4	65	35	62.990	221.50	0.270	23198	2.747
90	2	90	46	64.660	790.23	0.470	18467	1.548
90	3	90	47	64.998	530.84	0.316	27032	2.265
90	4	90	48	65.023	402.92	0.239	32113	2.691

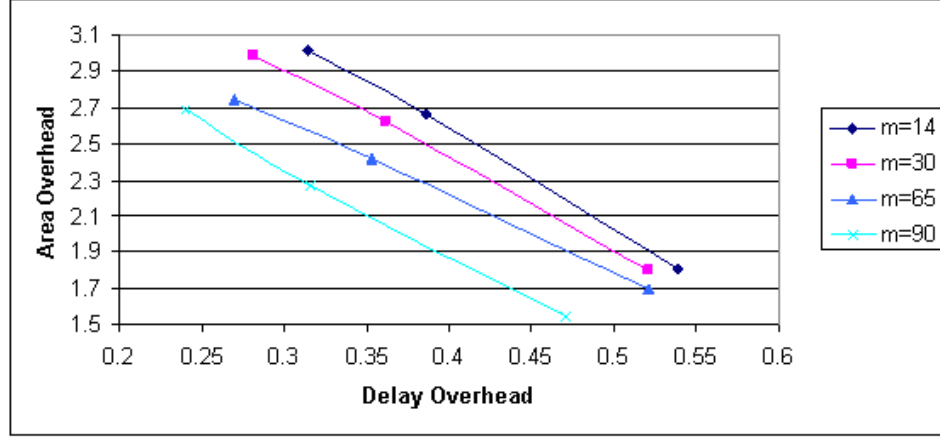


Figure 7.2: The ECC_{PS} Cryptoprocessor Delay and Area Overheads.

7.4 ECC_{SS} vs. ECC_{PS} Comparison

The lower bound on the area-time cost of a given design is usually employed as a performance metric $(\text{area}) \times (\text{time})^{2\alpha}$, $0 \leq \alpha \leq 1$, where the choice of α determines the relative importance of area and time [108]. Such lower bounds have been obtained for several problems, e.g., discrete Fourier transform, matrix multiplication, binary addition, and others [108]. Once the lower bound on the chosen performance metric is known, designers attempt to devise algorithms and designs which are optimal for a range of area and time values. Even though a design might be optimal for a certain range of area and time values, it is nevertheless of interest to obtain designs for minimum values of time, i.e., maximum speed performance, as well as designs for minimum area. In order to make a more meaningful comparison between the two secure cryptoprocessor architectures, both the AT and AT^2 measures are

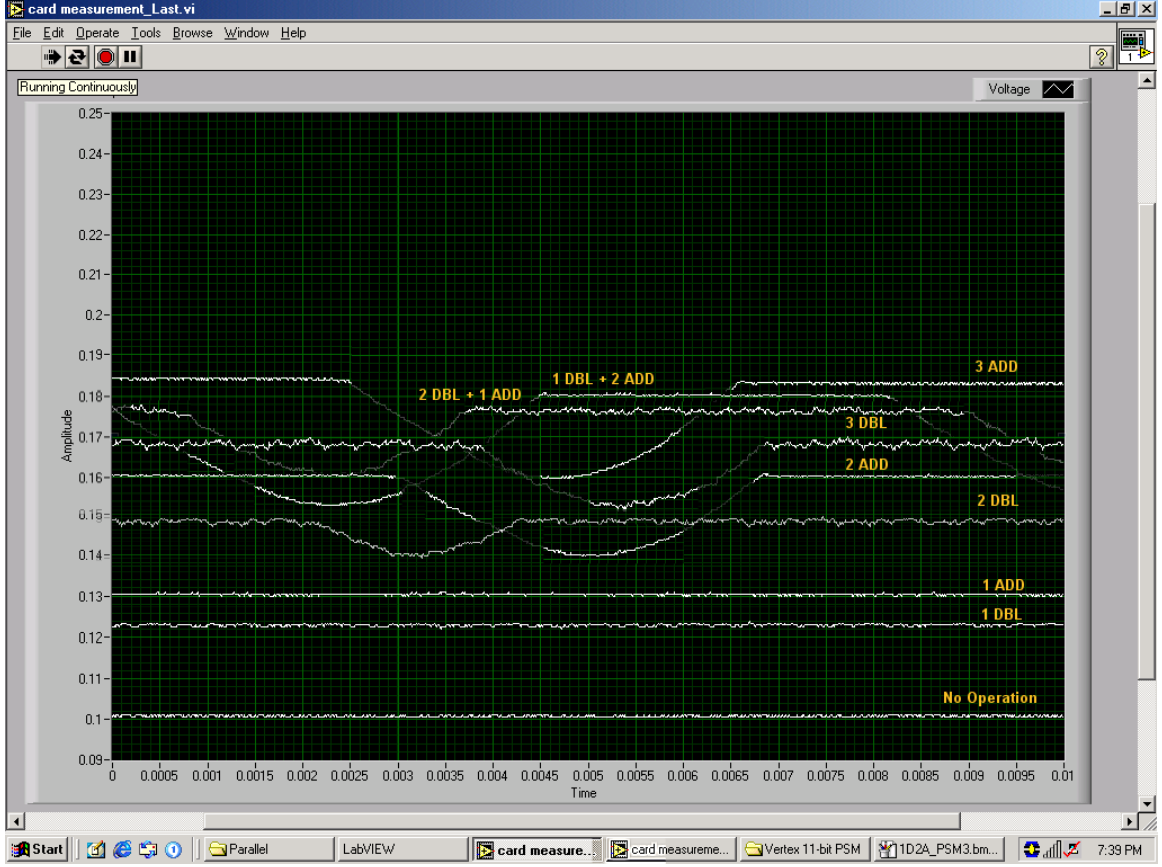


Figure 7.3: The ECC_{PS} Cryptoprocessor Voltage Trace with $m = 11$.

evaluated for both architectures.

The AT and AT^2 performance metrics were studied for both secure architectures. In one case, AT and AT^2 of both designs are studied for various key sizes with a fixed number of key partitions ($u = 4$) and in another case for a fixed key size ($m = 173$) and variable number of key partitions. For the first case, the number of key partitions is fixed at $u = 4$ while the values of $m = 14, 30, 65, 90$ and 173 were considered. Table 7.4 shows the normalized AT and AT^2 for the ECC_{SS}

and the ECC_{PS} cryptoprocessors for $u = 4$ key partitions since it provides more resistance against power analysis attacks. Figures 7.4 and 7.5 show the AT and AT^2 complexities respectively. The results shown in Figures 7.4 and 7.5 indicate that the ECC_{PS} cryptoprocessor has better AT and AT^2 performance metrics compared to the ECC_{SS} cryptoprocessor.

For the second case, the key size is fixed at $m = 173$ while several values for the number of key partitions ($u = 10, 20, 30, 40, 50$ and 60) were considered. The delay overhead of the ECC_{SS} and the ECC_{PS} cryptoprocessors can be approximated as:

$$ECC_{SS} \text{ Delay Overhead} = \frac{(m)DBL_{CC} + (0.7m + u - 1)ADD_{CC}}{(m)DBL_{CC} + \frac{m}{2}ADD_{CC}}$$

and

$$ECC_{PS} \text{ Delay Overhead} = \frac{(m)DBL_{CC} + \frac{m}{2}ADD_{CC}}{(m)DBL_{CC} + \frac{u}{2}ADD_{CC}}$$

where DBL_{CC} and ADD_{CC} are the required clock cycles for performing point doubling and point addition respectively. Tables 7.2 and 7.3 show that the normalized area overhead of the ECC_{SS} and the ECC_{PS} cryptoprocessors increases at an approximately rate of 5% and 50% respectively per each additional key partition. The AT and AT^2 measures over $GF(2^{173})$, with variable number of key partitions u , are depicted in Figures 7.6 and 7.7 respectively. The results of Figures 7.6 and 7.7 show that the ECC_{PS} cryptoprocessor also enjoys better AT and AT^2 with more key partitions than the ECC_{SS} cryptoprocessor.

Table 7.4: Area-Time Complexity Comparison for $u = 4$.

m	ECC_{ss}		ECC_{ps}	
	AT	AT ²	AT	AT ²
14	1.84	2.71	0.947	0.297
30	1.68	2.27	0.839	0.236
65	1.57	2.03	0.742	0.201
90	1.53	1.94	0.645	0.155

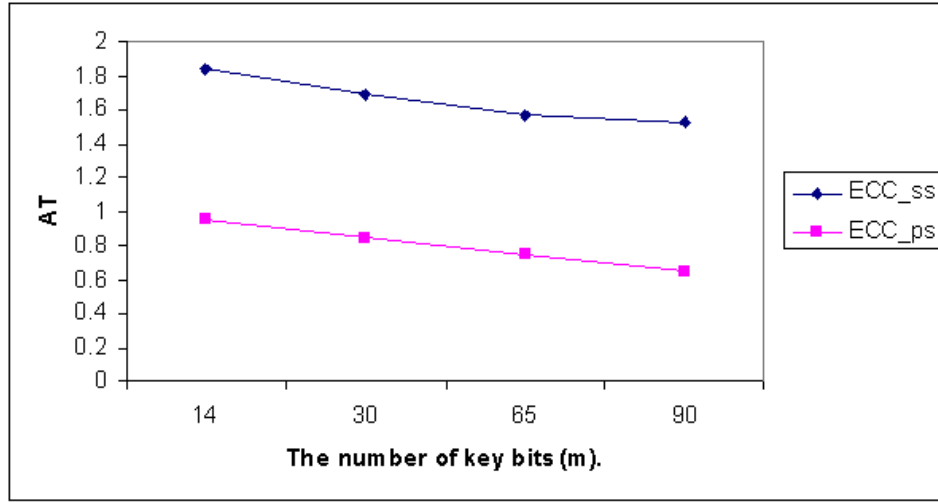


Figure 7.4: The Area-Time Complexity for $u = 4$.

7.5 Summary

In this chapter, three elliptic curve cryptoprocessor architectures for curves defined over $GF(2^m)$ have been modeled using VHDL. The developed VHDL models are parameterized to allow synthesizing the cryptoprocessors with different architectural features.

Synthesis results for the proposed secure sequential cryptoprocessor show that

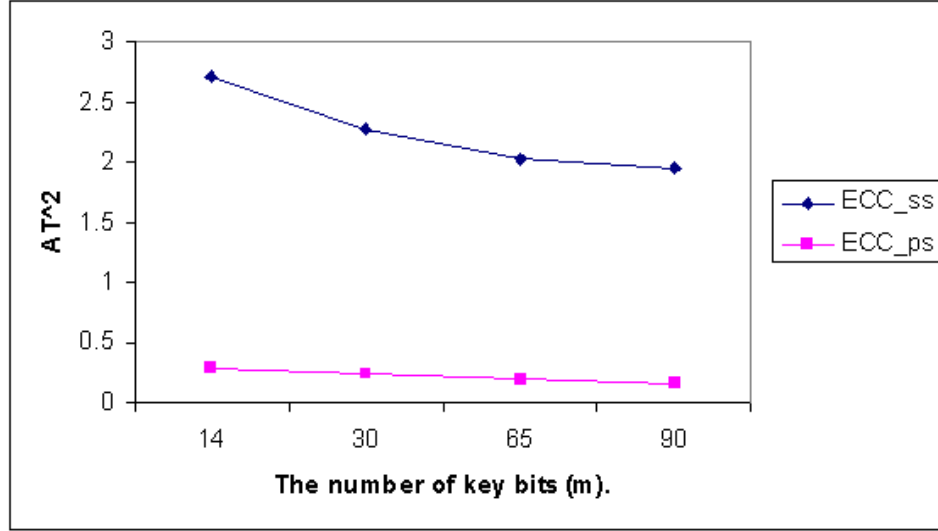


Figure 7.5: The Area-Time² Complexity for $u = 4$.

increasing the number of key partitions slightly increases the area. In contrast, increasing the number of key partitions for the proposed parallel cryptoprocessor significantly increases the space requirements since several scalar multipliers are employed.

The sequential cryptoprocessor requires more point additions than the reference non-secure cryptoprocessor so as to accumulate the results of the key partitions. For the parallel cryptoprocessor, however, a significant overall speedup is obtained since several scalar multipliers are used.

The AT and AT² performance measures for the proposed secure cryptoprocessors show that the parallel cryptoprocessor has better AT and AT² compared to the sequential cryptoprocessor.

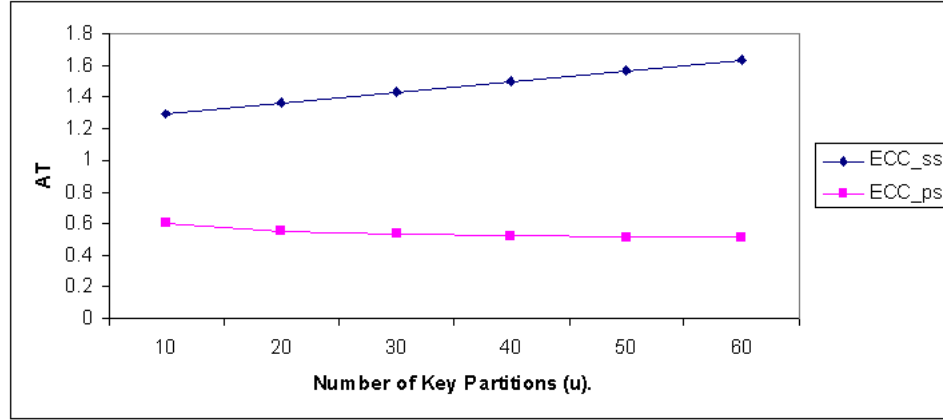


Figure 7.6: The Approximated Area-Time Complexity for $m = 173$.

The three cryptoprocessor has been implemented on a Xilinx FPGA and the voltage trace of the core logic of the parallel cryptoprocessor has been measured for different parallel point operations to demonstrate resistance against power analysis attacks. The results show that using the parallel cryptoprocessor significantly confuses leaked information without performing extra dummy point operations.

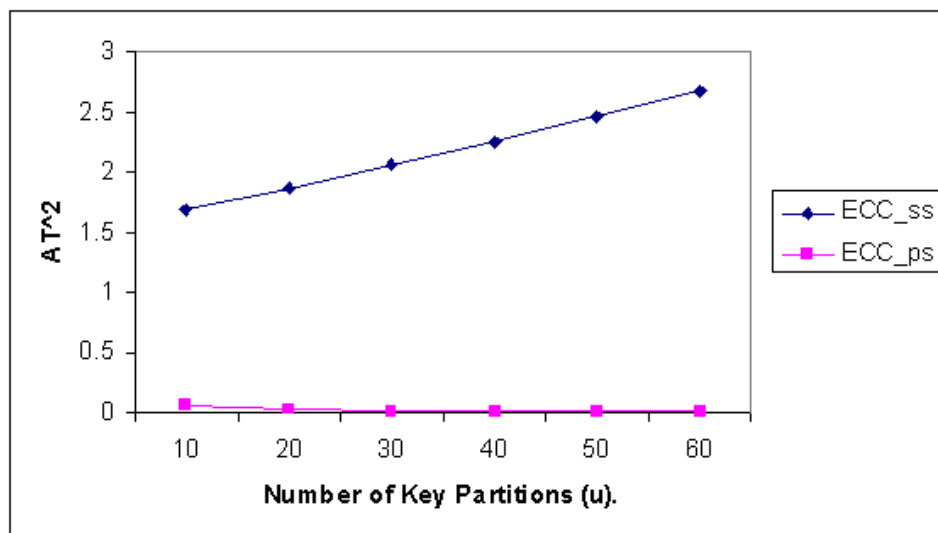


Figure 7.7: The Approximated Area-Time² Complexity for $m = 173$.

Chapter 8

Conclusions and Future Research

In this dissertation, three elliptic curve cryptoprocessor architectures for curves defined over $GF(2^m)$ have been modeled using VHDL. The developed VHDL models are parameterized to allow synthesizing the cryptoprocessors with different architectural features. The proposed architectures allow designers to tailor performance and hardware requirements according to their performance and cost objectives.

Two of these architectures are proposed as being secure against power analysis attacks, while the non-secure one has been used as a reference model for area and delay comparisons. The proposed power analysis attack resistant cryptoprocessors include one sequential and another parallel cryptoprocessor architectures. The sequential cryptoprocessor architecture uses multilevel resistance measures against power analysis attacks. These include resistance measures at the key level, the key partition level, and at the bit level.

At the key level, the key is divided into a number of partitions which are sequentially processed in a randomized order. The encoding of each key partition is randomly selected to be either in binary or in Non-Adjacent-Form (NAF) at the key partition level. Furthermore, at the key partition level, the direction of bit inspection for binary encoded key partitions is randomly assigned to be either most-to-least or least-to-most. Finally, the zeros, at the bit level, are randomized to appear sometimes as ones by performing dummy point additions. The sequential cryptoprocessor architecture certainly makes it very difficult for cryptanalysts to infer the key from leaked information in such multilevel resistance secure environment.

The proposed parallel cryptoprocessor provides high speed through using parallel scalar multipliers that operate in parallel on different key partitions. Parallel scalar multiplications of different key partitions are exploited as a countermeasure against power analysis attacks. Furthermore, the inspection order of each key partition is randomly decided to increase the immunity against power analysis attacks. The parallel cryptoprocessor does not need extra dummy operations as performed in the sequential processor. Hence, high performance gain is achieved while at the same time providing adequate resistance against power analysis attacks.

For the proposed secure sequential cryptoprocessor, synthesis results show that increasing the number of key partitions slightly increases the area. In contrast, increasing the number of key partitions for the proposed parallel cryptoprocessor significantly increases the space requirements since parallel scalar multipliers are

employed. The time complexity of the sequential cryptoprocessor requires more point additions than the reference non-secure cryptoprocessor so as to accumulate the results of various key partitions. However, for the parallel cryptoprocessor, a significant overall speedup is obtained.

Furthermore, the AT and AT^2 performance measures for the proposed secure cryptoprocessors have been evaluated. The results show that the proposed parallel cryptoprocessor has better AT and AT^2 compared to the proposed sequential cryptoprocessor.

In order to demonstrate resistance against power analysis attacks, the parallel cryptoprocessor has been implemented on a Xilinx FPGA and the voltage trace of the core logic has been measured for different parallel point operations. The results have shown that using the parallel cryptoprocessor significantly confuses leaked information.

Future research may further investigate the following:

1. Exploring the sequential-parallel mixed design.
2. Finding an optimal configuration with the best number of key partitions.
3. Evaluating both architectures on other ASIC platforms (e.g. standard cells).
4. Extending the same ideas to hyper-elliptic curves which require less number of bits (80-bits).

Bibliography

- [1] Koblitz, N., Elliptic curve cryptosystems. *Mathematics of Computation*, Vol. 48, pp. 203–209, 1987.
- [2] ANSI X9.62 – 1998, Public Key Cryptography for the Financial Services Industry: Curve Digital Signature Algorithm (ECDSA), 1998.
- [3] IEEE P1363, IEEE Standard Specifications for Public-Key Cryptography, 2000.
- [4] National Institute of Standards and Technology, Recommended Elliptic Curves for Federal Government Use, Appendix to FIPS 186–2, 2000.
- [5] Standards for Efficient Cryptography Group-Certicom Research, SEC 1: Elliptic Curve Cryptography, Version 1.0, 2000. <http://www.secg.org/>
- [6] Standards for Efficient Cryptography Group-Certicom Research, SEC 2: Recommended Elliptic Curve Cryptography Domain Parameters, Version 1.0, 2000.
- [7] Wireless Application Protocol (WAP) Forum, Wireless Transport Layer Security (WTLS) Specification. <http://www.wapforum.org/>.
- [8] FIPS 186-2. Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186–2, U.S. Department of Commerce/N.I.S.T. National Institute of Standards and Technology, 2000.
- [9] Rivest, R., Shamir, A. and Adleman, L., A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, Vol. 21, No.2, pp. 120–126, 1978.
- [10] El Gamal, T., A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *Advances in Cryptology: Proceedings of CRYPTO'84*, Springer Verlag, pp. 10–18, 1985.

- [11] Kocher, C., Jaffe, J. and Jun, B., Differential power analysis, CRYPTO '99, LNCS 1666, pp. 388–397, 1999.
- [12] Diffie, W. and Hellman, M., New directions in cryptography. IEEE Transactions on Information Theory, Vol. 22, pp. 644–654, 1976.
- [13] Schroepfel, R., Orman, H., O'Malley, S. and Spatscheck, O., Fast key exchange with elliptic curve systems. In Advances in Cryptology - CRYPTO'95, LNCS 963, pp. 43–56. Springer-Verlag, 1995.
- [14] Win, E., Bosselaers, A., Vandenberghe, S., Gerssem, P. and Vandewalle, J., A fast software implementation for arithmetic operations in $GF(2^n)$. In Advances in Cryptology - ASIACRYPT'96, LNCS 1163, pp. 65–76, Springer-Verlag, 1996.
- [15] Win, E., Mister, S., Preneel, B. and Wiener, M., On the performance of signature schemes based on elliptic curves. In Algorithmic Number Theory, Proceedings Third International Symposium, LNCS1423, pp. 252–266. Springer-Verlag, 1998.
- [16] Lopez, J. and Dahab, R., Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. In Cryptographic Hardware and Embedded Systems - CHES'99, LNCS 1717, pp. 316–327, Springer-Verlag, 1999.
- [17] Hankerson, D., Lopez, J. and Menezes, A., Software implementation of elliptic curve cryptography over binary fields. In Cryptographic Hardware and Embedded Systems - CHES'2000, LNCS 1965, pp. 1–24, Springer-Verlag, 2000.
- [18] Brown, M., Hankerson, D., Lopez, J. and Menezes, A., Software implementation of the NIST elliptic curves over prime fields. In Proc. CT-RSA, LNCS 2020, 2001, pp. 250–265.
- [19] Agnew, G., Mullin, R. and Vanstone, S., A fast elliptic curve cryptosystem. Adv. in Cryptology: EUROCRYPT'89, LNCS 434, pp. 706–708, 1989.
- [20] Agnew, G., Mullin, R. and Vanstone, S., An implementation of elliptic curve cryptosystems over F_2^{155} . IEEE Journal on Selected Areas in Communications, Vol. 11, No. 5, pp.804–813, 1993.
- [21] M. Rosner. Elliptic curve cryptosystems on reconfigurable hardware. Master's thesis, ECE Dept., Worcester Polytechnic Institute, Worcester, USA, May 1998.
- [22] Gao, L., Shrivastava, S. and Sobelman, G., Elliptic curve scalar multiplier design using FPGAs. In Cryptographic Hardware and Embedded Systems - CHES'99, LNCS 1717, pp. 257–268, Springer-Verlag, 1999.

- [23] Okada, S., Torii, N., Itoh, K. and Takenaka, M., Implementation of elliptic curve cryptographic coprocessor over $GF(2^m)$ on an FPGA. in Proc. Workshop on Cryptographic Hardware and Embedded Systems, LNCS 1965, 2000, pp. 25-40.
- [24] Goodman, J. and Chandrakasan, A., An energy efficient reconfigurable public-key cryptography processor architecture. Proc. Cryptographic Hardware and Embedded Systems, pp. 175-190, 2000.
- [25] Leung, K., Ma, K., Wong, W. and Leong, P., FPGA implementation of a microcoded elliptic curve cryptographic processor. In Eight Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM'2000, Napa Valley, California, USA, 2000.
- [26] Orlando, G. and Paar, C., A High-Performance Reconfigurable Elliptic Curve Coprocessor for $GF(2^m)$, In CHES'2000, LNCS 1965 , pp. 41–56, 2000.
- [27] Ernst, M., Klupsch, S., Hauck, O. and Huss, S., Rapid Prototyping for Hardware Accelerated Elliptic Curve Public Key Cryptosystems, Proc. 12th IEEE Workshop on Rapid System Prototyping (RSP01), Monterey, CA, 2001.
- [28] Smart, N., The Hessian form of an elliptic curve. In Proc. Workshop on Cryptographic Hardware and Embedded Systems, LNCS 2162, pp. 118-125, 2001.
- [29] Orlando, G., Efficient Elliptic Curve Processor Architectures for Field Programmable Logic, PhD dissertation, 2002.
- [30] Leong, P. and Leung, K., A Microcoded Elliptic Curve Processor Using FPGA Technology. IEEE Transactions on VLSI, Vol. 10, No. 5, 2002.
- [31] Ernst, M., Jung, M., Madlener, F., Huss, S. and Blmel, R., A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over $GF(2^n)$, Cryptographic Hardware and Embedded Systems - CHES'2002, LNCS 2523, pp. 381–399, 2002.
- [32] Gura, N., Eberle, H. and Shantz, S., Generic implementations of elliptic curve cryptography using partial reduction. Proc. ACM Computer and Communications Security, pp. 108-116, 2002.
- [33] Gura, N., Shantz, S., Eberle, H., Gupta, S., Gupta, V., Finchelstein, D., Goupy, E. and Stebila, D., An end-to-End systems approach to elliptic curve cryptography. Proc. Cryptographic Hardware and Embedded Systems, pp. 349-365, 2002.

- [34] Jung, M., Madlener, F., Ernst, E. and Huss, S. A reconfigurable coprocessor for finite field multiplication in $GF(2^m)$. In Proc. IEEE Workshop on Heterogeneous Reconfigurable System on Chip, 2002.
- [35] Kerins, T., Popovici, E., Marnane, W. and Fitzpatrick, P., Fully parameterizable elliptic curve cryptography processor over $GF(2^m)$. Proc. Field-Programmable Logic and Applications, pp. 750-759, 2002.
- [36] Bednara, M., Daldrup, M., Gathen, J., Shokrollahi, J. and Teich, J., Reconfigurable Implementation of Elliptic Curve Crypto Algorithms. The proceedings of IPDPS 2003.
- [37] Satoh, A. and Takano, K., A Scalable Dual-Field Elliptic Curve Cryptographic Processor. IEEE Transactions on Computers, Vol. 52, No. 4, 2003.
- [38] Lutz, J. and Hasan, A., High performance FPGA based elliptic curve cryptographic co-processor. In Proc. IEEE Conf. on Information Technology: Coding and Computing, pp. 486-492, 2004.
- [39] Mentens, N., rs, S. and Preneel, B., An FPGA implementation of an elliptic curve processor over $GF(2^m)$. Proc. GVLIS, pp. 454-457, 2004.
- [40] Jarvinen, K., Tommiska, M. and Skytta, J., A scalable architecture for elliptic curve point multiplication. In Proc. IEEE International Conference on Field-Programmable Technology, pp. 303-306, 2004.
- [41] Antola, A., Bertoni, G., Breveglieri, L. and Maistri, P., Parallel architectures for elliptic curve cryptoprocessors over binary extension fields. Proc. of the 46th IEEE International Midwest Symposium on Circuits and Systems MWS-CAS'03, Vol. 2, pp. 802-805, 2004
- [42] Saqib, N., Rodriguez-Henriquez, F. and Diaz-Perez, A., A Parallel Architecture for Fast Computation of Elliptic Curve Scalar Multiplication over $GF(2^m)$. Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS04), pp. 144-151, 2004.
- [43] Sozzani, F., Bertoni, G., Turcato, S. and Breveglieri, L., A parallelized design for an elliptic curve cryptosystem coprocessor. International Conference on Information Technology: Coding and Computing ITCC 2005, Vol. 1, pp. 626-630, 2005.
- [44] Batina, L., Mentens, N., Preneel, B. and Verbauwhede, I., Side-channel aware design: algorithms and architectures for elliptic curve cryptography over $GF(2^n)$. 16th IEEE International Conference on Application-Specific Systems, Architecture Processors ASAP, pp. 350-355, 2005.

- [45] Cheung, R., Telle, N., Luk, W. and Cheung, P., Customizable elliptic curve cryptosystems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 13, No. 9, pp. 1048–1059, 2005.
- [46] Cheung, R., Luk, W. and Cheung, P., Reconfigurable elliptic curve cryptosystems on a chip. *Proc. Design, Automation and Test in Europe*, Vol. 1, pp. 24–29, 2005.
- [47] Al-Somani, T. and Ibrahim, M., High Performance Elliptic Curve $GF(2^m)$ Cryptoprocessor Secure Against Timing Attacks, *International Journal of Computer Science and Network Security (IJCSNS)*, Vol. 6, No. 1B, 2006.
- [48] Al-Somani, T., Ibrahim, M. and Gutub, A., Highly Efficient Elliptic Curve Crypto-Processor with Parallel $GF(2^m)$ Field Multipliers, *Journal of Computer Science*, Vol. 2, No. 5, pp. 395–400, 2006.
- [49] Al-Somani, T., Ibrahim, M. and Gutub, A., High Performance Elliptic Curve $GF(2^m)$ Crypto-Processor. To appear in *Information Technology Journal*, Vol. 5, No. 3, 2006.
- [50] Guajardo, J. and Paar, C., Efficient algorithms for elliptic curve cryptosystems. In *Advances in Cryptology - CRYPTO'97*, LNCS 1294, pp. 342–356. Springer-Verlag, 1997.
- [51] Janssen, S., Thomas, J., Borremans, W., Gijssels, P., Verbauwhede, I., Vercauteren, F., Preneel, B. and Vandewalle, J., Hardware/Software Co-design of an Elliptic Curve Public-key Cryptosystem. In *Proceedings IEEE Workshop on Signal Processing Systems, SiPS-2001*, Antwerp, Belgium, pp. 209–216, 2001.
- [52] Shuhua, W. and Yuefei, Z., A Timing-and-Area Tradeoff $GF(p)$ Elliptic Curve Processor Architecture for FPGA. *Proc. International Conference on Communications, Circuits and Systems 2005*, Vol. 2, pp. 1308–1312, 2005.
- [53] Gutub, A., Merging $GF(p)$ Elliptic Curve Point Adding and Doubling on Pipelined VLSI Cryptographic ASIC Architecture, *International Journal of Computer Science and Network Security (IJCSNS)*, Vol.6, No.3A, pp. 44-52, March 2006.
- [54] Gutub, A., Fast 160-Bits $GF(p)$ Elliptic Curve Crypto Hardware of High-Radix Scalable Multipliers. *International Arab Journal of Information Technology (IA-JIT)*, to appear on January, 2007 in vol. 4, no. 1, 2007.
- [55] Biggs, N., *Discrete Mathematics*. Oxford University Press, New York, 1985.

- [56] McEliece, R., Finite Fields for Computer Scientists and Engineers. Kluwer Academic Publishers, 1987.
- [57] Lidl, R. and Niederreiter, H., Introduction to finite fields and their applications. Cambridge University Press, Cambridge, UK, revised edition, 1994.
- [58] Mullin, R., Onyszchuk, I., Vanstone, S. and Wilson, R., Optimal normal bases in $GF(p^m)$. Discrete Appl. Math., vol. 22, pp. 149–161, 1988/1989.
- [59] Rosing, M., Implementing Elliptic Curve Cryptography. Manning Publications Company, 1999.
- [60] Menezes, A., Elliptic Curve Public Key Cryptosystems. Kluwer Academic Publishers, 1993.
- [61] Cohen, H., Ono, T., and Miyaji, A., Efficient elliptic curve exponentiation using mixed coordinates. In Advances in Cryptology ASIACRYPT'98, Vol. 1514 of Lecture Notes in Computer Science, pp. 51–65, 1998.
- [62] Koyama, K. and Tsutsumi, Y., Speeding up elliptic cryptosystems by using signed binary window method. Advances in Cryptology Proc. Of Crypto'92, LNCS 740, Springer-Verlag, pp. 345–357, 1993.
- [63] Cohen, H., Miyaji, A. and Ono, T., Efficient elliptic curve exponentiation. Advances in Cryptology-Proc. Of ICICS'97, LNCS 1334, Springer-Verlag, pp. 282–290, 1997.
- [64] Lopez, J. and Dahab, R., Improved Algorithms for Elliptic Curve Arithmetic in $GF(2^n)$. SAC'98, LNCS 1556, pp. 201–212, Springer-Verlag, 1998.
- [65] Gordon, D., A Survey of Fast Exponentiation Methods. Journal of Algorithms, pp. 129–146, 1998.
- [66] Pollard, J., Monte Carlo methods for index computation mod p . Mathematics of Computation, Vol. 32, pp. 918–924, 1978.
- [67] Gallant, R., Lambert, R. and Vanstone, S., Improving the parallelized Pollard lambda search on binary anomalous curves. Math. Comp., Vol. 69, No. 232, pp. 1699–1705, 2000.
- [68] Menezes, A., Oorschot, P. and Vanstone, S., Handbook of Applied Cryptography. CRC Press, 1997.
- [69] Blake, I., Seroussi, G. and Smart, N., Elliptic Curves in Cryptography. Cambridge University Press, Cambridge, UK, First edition, 1999.

- [70] Lopez, J. and Dahab, R., An overview of elliptic curve cryptography. Technical Report IC-00-10, Institute of Computing, State University of Campinas, Campinas, Sao Paulo, Brazil, 2000.
- [71] Solinas, J., Improved algorithms for arithmetic on anomalous binary curves. Technical Report CORR-46, University of Waterloo, 1999.
- [72] Montgomery, P., Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, Vol. 48, pp. 243–264, 1987.
- [73] Brickell, E., Gordon, D., McCurley, K. and Wilson, D., Fast exponentiation with precomputation. In *Proc. Eurocrypt'92, Balatonfured*, 1992.
- [74] Lim, C. and Lee, P., More Flexibility Exponentiation with Precomputation. *Advances in Cryptology - Crypto'94, LNCS 839*, pp. 95–107, 1994.
- [75] Menezes, A. J., Blake, I. F., Gao, X., Mullin, R. C., Vanstone, S. A., And Yaghoobian, T., *Applications of Finite Fields*. Kluwer Academic Publishers, Boston, MA, 1993.
- [76] Al-Somani, T. and Amin, A., Hardware Implementations of $GF(2^m)$ Arithmetic using Normal Basis. To appear in *Journal of Applied Sciences* Vol. 6, No. 6, 2006.
- [77] Massey, J. L. and Omura, J. K., Computational method and apparatus for finite field arithmetic. US Patent No. 4587627, 1986.
- [78] Wang, C. C., Truong, T. K., Shao, H. M., Deutsch, L. J., Omura, J. K., And Reed, I. S., VLSI architectures for computing multiplications and inverses in $GF(2^m)$. *IEEE Trans. Comput.*, Vol. 34, No. 8, pp. 709–716, 1985
- [79] Hasan, M. A., Wang, M. Z., And Bhargava, V. K., A modified Massey-Omura parallel multiplier for a class of finite fields. *IEEE Trans. Comput.*, Vol 42, No. 10, pp. 1278–1280, 1993.
- [80] Gao, L. And Sobelman, G. E., Improved VLSI designs for multiplication and inversion in $GF(2^m)$ over normal bases. In *Proceedings of 13th Annual IEEE International ASIC/SOC Conference*. pp. 97–101, 2000.
- [81] Reyhani-Masoleh, A. And Hasan, M. A., Efficient digit-serial normal basis multipliers over binary extension fields. In *ACM Trans. on Embedded Computing Systems*, Vol. 3, No. 3, pp. 575–592, 2004.
- [82] Reyhani-Masoleh, A. And Hasan, M. A., A new construction of Massey-Omura parallel multiplier over $GF(2^m)$. *IEEE Trans. Comput.*, Vol. 51, No. 5 , pp. 511–520, 2002.

- [83] Koc, C. K. And Sunar, B., Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields. *IEEE Trans. Comput.*, Vol. 47, No. 3, pp. 353-356, 1998.
- [84] Sunar, B. And Koc, C. K., An efficient optimal normal basis Type II multiplier. *IEEE Trans. Comput.*, Vol. 50, No. 1, pp. 83-88, 2001.
- [85] Wu, H., Hasan, A., Blake, I., and Gao, S., Finite Field Multiplier Using Redundant Representation. *IEEE Trans. Comput.* vol. 51, No. 11, pp. 1306-1316, 2002.
- [86] Gao, S. and Vanstone, S., On Orders of Optimal Normal Basis Generators. *Math. Computation*, Vol. 64, No. 2, pp. 1227-1233, 1995.
- [87] T. Itoh and S. Tsujii, A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Info. Comput.*, Vol. 78, No.3, pp. 171-177, 1988.
- [88] G.L. Feng, A VLSI Architecture for Fast Inversion in $GF(2^m)$. *IEEE Trans. Computers*, Vol. 38, No. 10, pp. 1383-1386, 1989.
- [89] Takagi, N., Yoshiki, J. and Takagi, K., A Fast Algorithm for Multiplicative Inversion in $GF(2^m)$. Using Normal Basis. In *IEEE Trans. Computers*, Vol. 50, No. 5, 2001.
- [90] S.T.J. Fenn, M. Benaissa and D. Taylor, Fast normal basis inversion in $GF(2^m)$. *Electronics Letters*, Vol. 32, No. 17, 1996.
- [91] Calvo, I. and Torres, M., Complexity of the inversion in $GF(2^m)$. *Electronics Letters*, Vol. 33, No. 3, 1997.
- [92] Yen, S., Improved normal basis inversion in $GF(2^m)$. *Electronics Letters*, Vol. 33, No. 3, 1997.
- [93] Kocher, C., Timing Attacks on Implementations of Diffe-Hellman, RSA, DSS, and Other Systems. *CRYPTO'96*, LNCS 1109, pp. 104-113, 1996.
- [94] Goubin, L., A refined power-analysis attack on elliptic curve cryptosystems. In *Public Key Cryptography - PKC'03*, LNCS 2567, pp. 199-210, Springer-Verlag, 2003.
- [95] Akishita, T. and Takagi, T., Zero-value point attacks on elliptic curve cryptosystem. In *Information Security Conference - ISC'03*, LNCS 2851, pp. 218-233, Springer-Verlag, 2003.

- [96] Fouque, P. and Valette, F., The doubling attack - why upwards is better than downwards. In *Cryptographic Hardware and Embedded Systems - CHES'03*, LNCS 2779, pp. 269–280, Springer-Verlag, 2003.
- [97] Coron, J., Resistance against differential power analysis for elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems - CHES'99*, LNCS 1717, pp. 292–302, Springer-Verlag, 1999.
- [98] Joye, M. and Tymen, C., Protections against differential analysis for elliptic curve cryptography. *CHES'2001*, LNCS 2162, pp. 377–390, 2001.
- [99] Ha, J. and Moon, S., Randomized signed-scalar multiplication of ECC to resist power attacks. In *Cryptographic Hardware and Embedded Systems - CHES'02*, LNCS 2523, pp. 551–563, Springer-Verlag, 2002.
- [100] Smart, N., An analysis Goubin's refined power analysis attack. *Proc. of Cryptographic Hardware and Embedded Systems - CHES'03*, LNCS 2779, pp. 281–290, Springer-Verlag, 2003.
- [101] Mamiya, H., Miyaji, A. and Morimoto, H., Efficient countermeasure against RPA, DPA, and SPA. In *Cryptographic Hardware and Embedded Systems - CHES'04*, LNCS 3156, pp. 343–356, Springer-Verlag, 2004.
- [102] Messerges, T., Dabbish, E. and Sloan, R., Investigations of Power Analysis Attacks on Smartcards. Preprint, *USENIX Workshop on Smartcard Technology*, 1999.
- [103] Itoh, K., Izu, T. and Takenaka, M., Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA. *Cryptographic Hardware and Embedded Systems: Proceedings of CHES'2002*, LNCS 2523, Springer-Verlag, pp. 129–143, 2002.
- [104] May, D., Muller, H. and Smart, N., Random Register Renaming to Foil PA. *CHES'2001*, LNCS 2162, pp. 28–38, Springer-Verlag, 2001.
- [105] Itoh, K., Izu, T. and Takenaka, M., A Practical Countermeasure against Address-Bit Differential Power Analysis. *Cryptographic Hardware and Embedded Systems: Proceedings of CHES'2003*, LNCS 2779, Springer-Verlag, pp. 382–396, 2003.
- [106] Kobitz, N., Menezes, A. and Vanstone, S., The State of Elliptic Curve Cryptography. *Designs, Codes and Cryptography*, Vol. 19, No. 2–3, pp. 173–193, 2000.

- [107] Joye, M. and Tymen, C., Compact Encoding of Non-Adjacent Forms with Applications to Elliptic Curve Cryptography. Public Key Cryptography, vol. 1992 of Lecture Notes, in Computer Science, pp. 353–364, Springer-Verlag, 2001.
- [108] Thompson, D., A complexity theory for VLSI. Ph.D. dissertation, Carnegie Mellon University, Dep. Computer Science, 1980.

Vita

- Turki Faisal Al-Somani.
- Born in Taif, Saudi Arabia, on July 07, 1974.
- Completed Bachelor of Science in Electrical and Computer Engineering from King Abdul-Aziz University, Jeddah, Saudi Arabia, in January 1997.
- Completed Masters of Science in Electrical and Computer Engineering from King Abdul-Aziz University, Jeddah, Saudi Arabia, in May 2000.
- Email: tfsomani@uqu.edu.sa.